A Robust Approach to Predict the Popularity of Songs by Identifying Appropriate Properties

by

S. Anjana

2015/CS/013

This dissertation is submitted to the University of Colombo School of Computing

In partial fulfillment of the requirements for the

Degree of Bachelor of Science Honours in Computer Science

University of Colombo School of Computing

35, Reid Avenue, Colombo 07,

Sri Lanka

July 2020

Declaration

I, S. Anjana (2015/CS/013) hereby certify that this dissertation entitled "A Robust Approach to Predict the Popularity of Songs by Identifying Appropriate Properties" is entirely my own work and it has never been submitted nor is currently been submitted for any other degree.

....

Date S. Anjana

I, Dr. K. L. Jayaratne, certify that I supervised this dissertation entitled "A Robust Approach to Predict the Popularity of Songs by Identifying Appropriate Properties" conducted by S. Anjana in partial fulfillment of the requirements for the degree of Bachelor of Science Honours in Computer Science.

....

Date Dr. K. L. Jayaratne

Abstract

Hit Song Science is a major research topic which is being discussed today in the field of Music Information Retrieval. In order to identify and predict whether a song could be a hit song or not is yet a challenging task.
This thesis investigates the ability of using machine learning to make predictions on Sinhala songs whether they would be a hit or not. More than 13,000 Sinhala songs were collected by web scraping in a popular Sinhala music website which also contributes by presenting a dataset that can be used for further research purposes. The number of downloads and the view counts were used to derive a equation to measure the popularity. The features extracted of each song is used by the XGBoost classification algorithm. The songs are initially grouped into 3 classes based on their popularity, and later by performing machine learning algorithms on a set of features extracted on each song, the impact of the Linear Predictive Coding (LPC) overall average (LOA) and Mel-Frequency Cepstral Coefficients (MFCC) features, to the end result is indicated by the use of SHAP process.

Preface

The dataset construction using techniques such as web scraping and feature extraction of the music clips were scripted by myself. Also, the results obtained using the machine learning algorithms which are stated in this research are carried out by myself itself.

Acknowledgement

I would like to express my very great appreciation to Dr. K.L. Jayaratne, my research supervisor for his valuable guidance, feedback, support and encouragement throughout this research work.

I would also like to give special thanks to my co-supervisor Dr. Kasun Gunawardana for his valuable and constructive suggestions during the planning and development of this research work and for his patient guidance and assistance in keeping my progress on schedule. Also his willingness to give his time so generously has been very much appreciated.

My special thanks are extended to the staff of Outstanding Song Creators Association (OSCA) for their assistance with guiding me about the music copyright related information, and the owners of the sarigama.lk website for having the song related statistics publicly and freely available.

I wish to acknowledge the help provided by my examiners Dr. H. A. C. Kaldera and Dr. K.K. Karunanayaka for their valuable feedback in every evaluation to help me improve my work.

Finally, I wish to thank my beloved parents for their support, encouragement and being there for me in all my hardships.

Table of Contents

List of Figures

List of Tables

List of Acronyms

BLOB Binary Large Object

ERD Entity Relationship Diagram

FN False Negative

FP False Positive

HTML Hypertext Markup Language

HTTP HyperText Transfer Protocol

JAR Java ARchive

LOA LPC overall average

LPC Linear Predictive Coding

MFCC Mel-Frequency Cepstral Coefficients

MP3 Moving Picture Experts Group (MPEG) Audio Layer-3

MPEG Moving Picture Experts Group

TN True Negative

TP True Positive

URL Uniform Resource Locator

WAV Waveform Audio File Format

Chapter 1

Introduction

1.1 Background to the Research

The music industry is one of the most widely spread industries around the world. The expansion of online streaming services has enabled artists to promote their musical productions across the world, and many newcomers to the industry were lately identified. It has become an easy task for an artist to promote their musical work online to the same audience, regardless of their skill level. But it would be doubtful whether these songs would become an international hit or go underrated/unpopular just after the release. Compared to the number of songs released every year, only a few numbers of songs would get listed as top hits.

When considering the music industry in Sri Lanka, there are currently thousands of songs produced throughout history. In addition to that, this number gradually increases every year. Several artists produce songs every year and, yet it would be doubtful whether these songs become popular or not. So, most of the artists would tend to use social media and other kinds of advertising platforms to promote their songs, which would temporarily make it a trending hit.

1.2 Research Problem and Research Questions

1.2.1 Research Problem

Some songs go viral, and some don't. In the modern-day, due to competitive reasons, song creators tend to use paid promotions to promote their songs to a much broader audience. The benefit of doing this is to obtain sponsorship for music releases and such similar reasons. However, if the song doesn't appear to be good for some reason, it wouldn't stay popular for a very long time. Once the promotion ends, the song's existence would even be forgotten by almost everyone. Meanwhile, some songs are still being enjoyed, which were produced years ago.

Apparently, these songs can be considered as all-time hits.
Some researches argue that the popularity of a song is defined as a totally random coincidence [2], while, others argue that these songs have some hidden qualities that lie within them which makes them popular for such a long time [3]. If these properties are identified and analyzed, it would be the key to predict the popularity of a song at the time of release. Currently, a vast amount of open-source datasets about songs have been collected for the purpose of research [4], and there is a number of researches conducted under this domain, where most of them have analyzed the high-level metadata of songs in order to predict its popularity [5]. And yet it is said to be uncertain if it is even possible to predict, better than random if a song will be popular or not [2].
This research project analyzes the low-level features of songs, such as the audio waveform and audio spectrum using machine learning algorithms and develop a tool to test a song's ability to be popular.

1.2.2 Research Questions

The main research questions which are being addressed within this research would be as following.

1. Do most of the popular songs have some qualities in common?
2. Does the audio signal related properties of a song be useful for predicting its popularity?
3. To what extent can popular songs be predicted using audio signal related properties?

1.2.3 Project Aim

This aim of this research is using machine learning algorithms on music in order to determine the qualities of popular songs and analyze their impact on popularity prediction of songs.

1.2.4 Objectives

The following objectives were accomplished when addressing the above research questions.

1. Identifying the audio signal characteristics of songs.
2. Identifying the popularity classes of each song.
3. Classifying the songs based on their audio signal related characteristics.

4. Identifying the relationship between audio signal related features and the popularity of the song.

1.3 Significance of the project

Using machine learning to predict a song's popularity, which is also known as "Hit Song Science," is a problem that has been around since the last decade. Many private organizations in the music industry are working on this problem, but the details about the successful results are kept confidential due to competitive reasons.

This research explores new methods of predicting a song's popularity using scientific methods, such as analyzing the audio waveform and correlating it with the popularity of the song, with the use of machine learning algorithms. The conclusions of this research project can be beneficial for many in the music industry. The results of this research can be used by composers to improve their artwork, and as a tool for the beginners and learners in the music industry, and other interested third parties can also request to purchase the intellectual property right of a song at the time of release.

1.4 Methodology

The research initially defines a measure of popularity, based on the data collected using web scraping. The songs are categorized according to this popularity measure and the audio waveform and audio frequency data is then analyzed using machine learning algorithms. Fourier Analysis and MFCC extraction is performed on the audio waveform for further processing. Classification using machine learning is used on the extracted features in order to co-relate with the popularity.
The research would next analyze the features that most impact the popularity of songs. This is considered to be helpful when predicting the popularity of a song
The process of this research can be divided into the following steps.

1. Data Collection
2. Preprocessing
3. Feature Extracting
4. Determining the Popularity
5. Model Training and Evaluation

Figure 1.1 depicts the approach of the research according to the above mentioned steps.

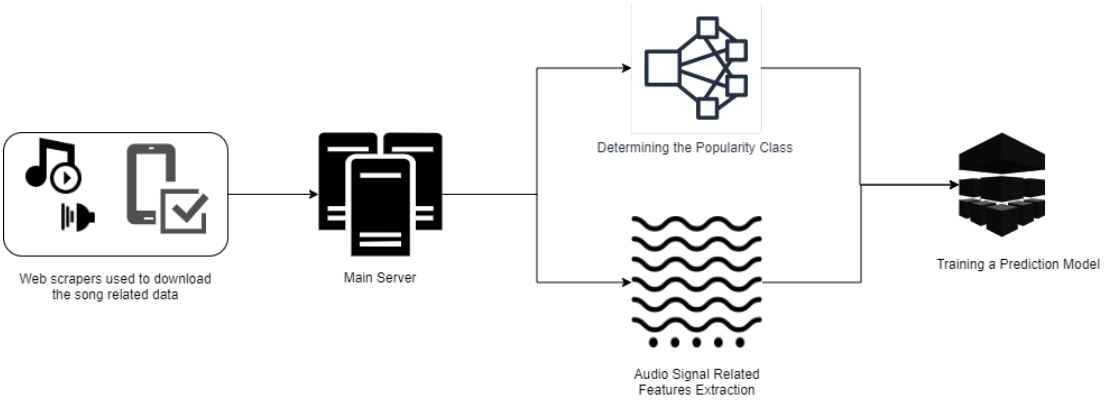

Figure 1.1: Research Approach

Data Collection

The Python *BeautifulSoap* and *requests* libraries were being used to collect data from a popular Sinhala song listing website. The following data items were being captured and stored.

1. Song Title
2. Authors (Singers/Musicians/Lyricists)
3. Download counts
4. View counts
5. MP3 audio files

1.4.1 Preprocessing

The fetched data was next being stored in a relational database for further processing. Also the features that were extracted from the audio files were stored in a separated table and treated as the feature vector when used to be provided as the input of the machine learning algorithm.

Feature Extraction

A number of features are extracted and when the number of dimensions are too large, a statistical approach such as considering the average and standard deviation of the coefficients were used. Some features related to the MFCC and LPC are presented with 10 and 13 parallel dimensions which are numbered according to the index.

1.5 Outline of the Dissertation

The first chapter provides an introduction to the research. The 2nd chapter is about literature reviews on related research. The 3rd chapter provides a design overview on the research. The implementation is described in the 4th chapter of this dissertation. The results obtained and the evaluation is described in the 5th chapter. The final 6th chapter includes the conclusions.

1.6 Scope and Delimitations

1.6.1 In Scope

This research project would intend to,

- Determine a measure for the popularity of songs.
- Analyze the audio signal related characteristics in both popular and unpopular songs.
- Correlate these features along with their popularity classes

1.6.2 Out Scope

This research project shall not,

- Predict estimated reach of songs based on the popularity.
- Be used as a personal music recommendation system.
- Analyze the activities of the brain in response to music frequencies

Chapter 2

Literature Review

2.1 Literature review

While Machine Learning algorithms are used in many fields for prediction problems, using it to predict the popularity of a song is a problem that has been addressed in many previous attempts. Several amounts of researches have been conducted on song popularity prediction, which is resulting in different conclusions.

Most of these researches are based on the Million Song Dataset (MSD), which is an open dataset provided by the Echo Nest API (Currently acquired by Spotify). The MSD is an attempt to help researchers by providing a large-scale dataset [4].

In 2008, an endeavor to validate the hypothesis that the popularity of songs can be predicted from acoustic or human features which were conducted by Pachet and Roy. Their experiment was based on a dataset of 32000 titles and 632 features, which was a considerably huge dataset at that time. However, they were not able to develop a good classification model and concluded that the popularity of a song could not be predicted by using state-of-the-art machine learning techniques [6].

In a 2011 study made by Ni et al., provided with a positive result on the problem of predicting music popularity. Their goal was to distinguish the top 5 hits from top 30-40 hit list. Their dataset was based on UK charts during a time period of 50 years. 5947 unique songs were collected from the Official Charts Company (OCC), and the audio features were extracted from The Echo Nest. They indicated that it is possible to identify music hits [7].

Another study conducted in the same year of 2011, by Borg and Hokkanen investigated if they could predict the popularity of a song based on its audio features and Youtube view counts. The features for audio tracks were obtained from The Echo Nest. For this task, several Support Vector Machines (SVM) were built

and achieved a very modest result. The SVMs, regardless of feature choice and parameters, never achieved more than 53% accuracy. They concluded that audio features alone do not seem to be good predictors of what makes a song popular. They suggest that popularity is likely driven by social forces [8].

Fan and Casey at the Dartmouth College compared the prediction of UK hit songs against Chinese hit songs. Their research, which was published in the year of 2013, also used a time-weighted Linear Regression model and SVMs on the audio features obtained using the Echo Nest API. The set of song tracks were collected from OCC for UK hits and ZhongGuoGeQuPaiHangBang for Chinese hits. They also defined a "Hit song" as the songs which were ranked 1-20 of the chart and a "non-hit song" as the songs which ranked from 21-40. Their research concluded that Chinese hit song prediction was more accurate than the UK hit song prediction. The error rate for predicting Chinese songs was 41%, while the prediction of UK hits generated a 52% error rate. The Chinese hits appeared to have significantly different characteristics than UK hits [9].

Herremans et al. focused on classifying dance hit songs in a 2014 study. They also extracted features using The Echo Nest API. The datasets used in the models were based on the OCC listings. The peak chart position of the songs which ranked from 1-40 was used in order to determine if a song was a dance hit or not. And they were able to create a dataset of dance hit songs from 2009 to 2013. Many Machine Learning algorithms such as Decision trees, Naive Bayes, Logistic Regression, and SVMs were used in their research. They concluded that the Logistic Regression was the best algorithm, which could be used to predict dance hit songs, by analyzing audio features with an accuracy of 0.65 [10].

Another study by Pham et al., at Stanford University in 2016, used different machine learning algorithms such as SVMs, Artificial Neural Networks, and Logistic Regression in order to test their ability to predict the popularity of music tracks. They classified using both audio features as well as metadata, which was obtained by the MSD. A subset of 2,717 tracks was used after removing records which consist of incomplete data that lacked some features of the initially selected 10,000 music tracks. They considered a hit song as a song with a high popularity value provided by The Echo Nest API. They finally concluded that all the models were performing with nearly similar accuracies, around 75% [5].

While most of these researches are preliminary based on western music tracks, there seems to be a lack of research conducted in the context of Sinhala songs. A recent study by Paranagama et al., in 2017 came up with a solution to automate

the process of determining the user ratings of songs by using a multilayer neural network. They've finally been able to achieve an accuracy of 50% in performance indices, with optimizing the code and solution in various ways such as using clustering to determine the labels and using pre-stored data for feeding the input [11].

Also, when considering music recommendation systems, Bo Shao et al. discusses the "Collaborative filtering" and "Content-based" approaches and their disadvantages. That is the initial startup problem of not having enough data on user preferences over music tracks and how it would negatively affect the recommender system's accuracy. They propose a novel approach to overcome these issues by using user access patterns along with content-based features [12].

Music recommendation is vastly used in industry-wide applications. According to Covington et al., YouTube uses the watch time, and the click-through rate in order to determine the trending videos. That is the number of users who clicked on a video thumbnail out of the total number of impressions, and the proportion of the total length of the video, that user watched has a significant impact rather than the view count of a video [13]. According to Zannettou et al., YouTube content creators tend to use more click baits with false information because the YouTubes algorithm does not consider such click baits in their recommendation algorithm [14].

HY Chang et al. implies how crucial the selection of appropriate music genre, when using music recommendations for stress-related therapies. They proposed a personalized stress-relieving music recommendation system based on EEG feedback [15].

Several previously conducted researches suggest that the popularity of a song can be predicted using machine learning techniques; some also disagree with it. Also, by considering the need of having studies on predicting the popularity of Sinhala songs, this research would expect to continue to investigate this problem.

Heckard et al. askes [16] the question: **"Will you always like the music that you like now?"**, and describes it as if you are about 20 years old, the likely answer is "yes," according to research reported in the Journal of Consumer Research [1]. The researchers concluded that we tend to acquire our popular music preferences during late adolescence and early adulthood.

In the study, 108 participants from 16 to 86 years old listened to 28 hit songs that had been on Billboard's Top 10 list for popular music sometime between 1932 and 1986. Respondents rated the 28 songs on a 10-point scale, with 1 corresponding to "I dislike it a lot" and 10 corresponding to "I like it a lot." Each individual's

ratings were then adjusted so that the mean rating for each participant was 0. On this adjusted rating scale, a positive score indicates a rating that was above average for a participant, whereas a negative score indicates a below-average rating. For each of the 108 participants 3 28 songs, a "song-specific age" was calculated representing how old the participant was when that song was popular. If the song was popular before the person was born, the song-specific age was negative. For example, the youngest participant in the study was born in 1971, so the song from 1932 was popular 39 years before that person was born, for a song-specific age of -39. The oldest participant was born in 1901, so the song from 1986 was popular 85 years after that person was born, for a song-specific age of +85. These were the two extremes, so the song-specific ages range from -39 to +85. Figure 3.3 shows the relationship between the average adjusted song ratings and the song-specific ages. There are 124 points in the scatter plot, one for each songspecific age from -39 to +85. The overall pattern in Figure 2.1 looks somewhat like an inverted U, and the highest preference ratings occur when song-specific ages are in the late teens and early twenties. A straight line does not describe the overall pattern, so the association is called nonlinear or curvilinear.

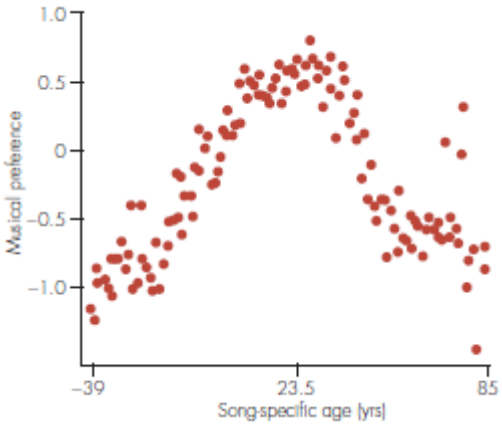

Figure 2.1: Song-specific age and music-preference score [1]

Review

For further processing, the understanding of how audio is being represented in a digital format is important.
The Sound is represented in the form of an audio signal having parameters such as frequency, bandwidth, decibel etc. A typical audio signal can be expressed as a function of Amplitude and Time as depicted in Figure 2.2.

These sounds are available in many formats which makes it possible for the computer to read and analyze them. In this research, the following audio formats

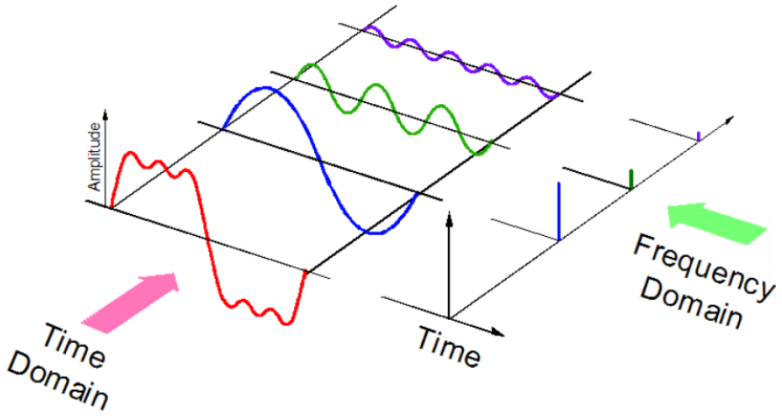

Figure 2.2: Representation of Audio Signals

will be used.

1. MPEG Audio Layer-3 (MP3) format
2. Waveform Audio File Format (WAV) format

Since the MP3 format uses lossy data-compression to encode data using inexact approximations and the partial discarding of data [1], it should be converted to the WAV format before the feature extraction begins. Since the WAV files are uncompressed, they contain lossless audio [2] that would occupy more disk space when storing.

[1] https://en.wikipedia.org/wiki/MP3
[2] https://www.howtogeek.com/392504/what-are-wav-and-wave-files-and-how-do-i-open-them/

Chapter 3

Design

3.1 Design

This research is designed to identify the audio signal related characteristics of songs, determining a measure for the popularity of songs and correlating them in order to observe a classification model.

Web scrapers are used to gather publicly available data from a popular Sinhala songs website[1]. The song title, artist related data, visits count, download count, and the song file in MP3 format is fetched using the Python script. Othe than the song file being saved as a Binary Large Object (BLOB), all the other song related metadata which were fetched is saved in the MySQL database in a relational form. Since a song can have many artists with many roles such as multiple vocalists, multiple lyricists, and multiple music composers while at the same time, a single artist also may contribute to multiple songs in may roles (types), a many-to-many (m : n) relationship should be implemented between the *songs* and *artists* entities. When normalized a new entity named *song_artist* is formed in order to hold the relationship. The composition of all three attributes which are *song_id*, *artist_id*, and *role* will be unique in the *song_artist* table. The Entity Relationship Diagram (ERD) in Figure 3.1 is part of the database design where the web-scraped data is stored.

The *download_count* and the *view_count* in the *songs* tables is stored for the purpose of ranking songs where a measure for the popularity could be determined. The song files are stored separately in the hard drive instead of being stored as a BLOB file in the database, where only the file name has to be recorded in the *songs* table. Since the song files are stored in the MP3 format, they are required to be converted in to the WAV in order to be further processed in the feature extraction stage. Usually the WAV files occupy reasonably more storage space in the hard drive. Therefore,

[1]https://sarigama.lk

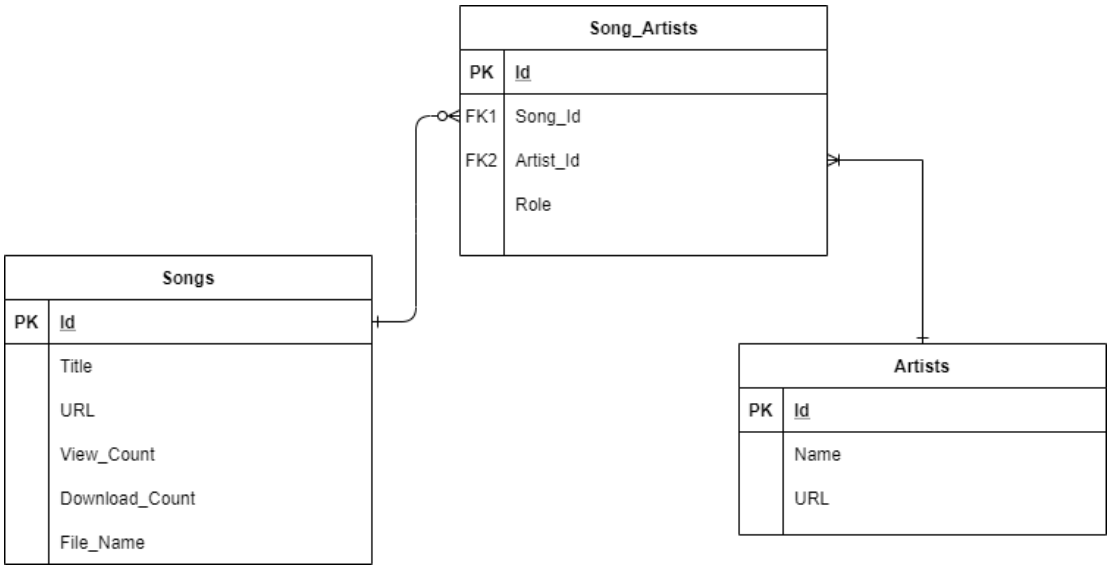

Figure 3.1: The partial ER Diagram which is used to store the web-scraped data

the WAV files are created temporarily, and as soon as the features are extracted, they are deleted in order to preserve the disk space.

The Figure 3.2 depicts the flow of the processes in this research, where initially a set of songs which were web-scraped and the audio features were extracted. The set of features were then being used as an input to the XGBoost classification in order to determine the popularity class of each of them. The popularity class is a factor derived by using the view count and the download count of each song.

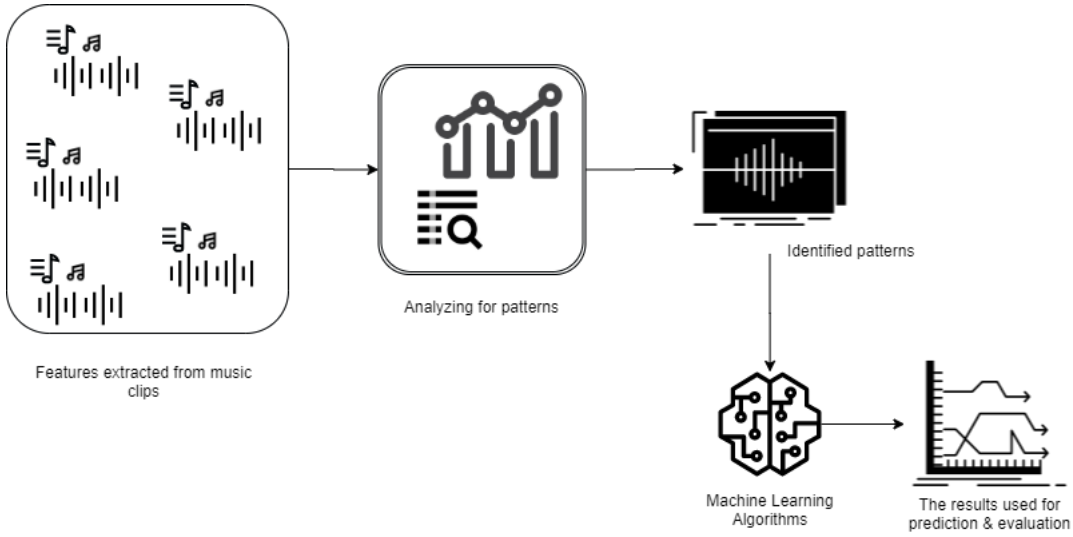

Figure 3.2: The process flow of the research

3.2 Tools and Programs Used

The Python **requests**[2] and **BeautifulSoap**[3] libraries were used to fetch the song related data and the song audio clips. The Python Librosa library [17] was used to extract the duration and tempo (beats per second) of each song. The jAudio tool [18] was being used for the extraction of all the other features.

The data collected was stored in a MySQL database. Figure 3.3 depicts the database design which was used for further processing using machine learning algorithms. This includes the table *features* which is being used to store the feature vector of each song.

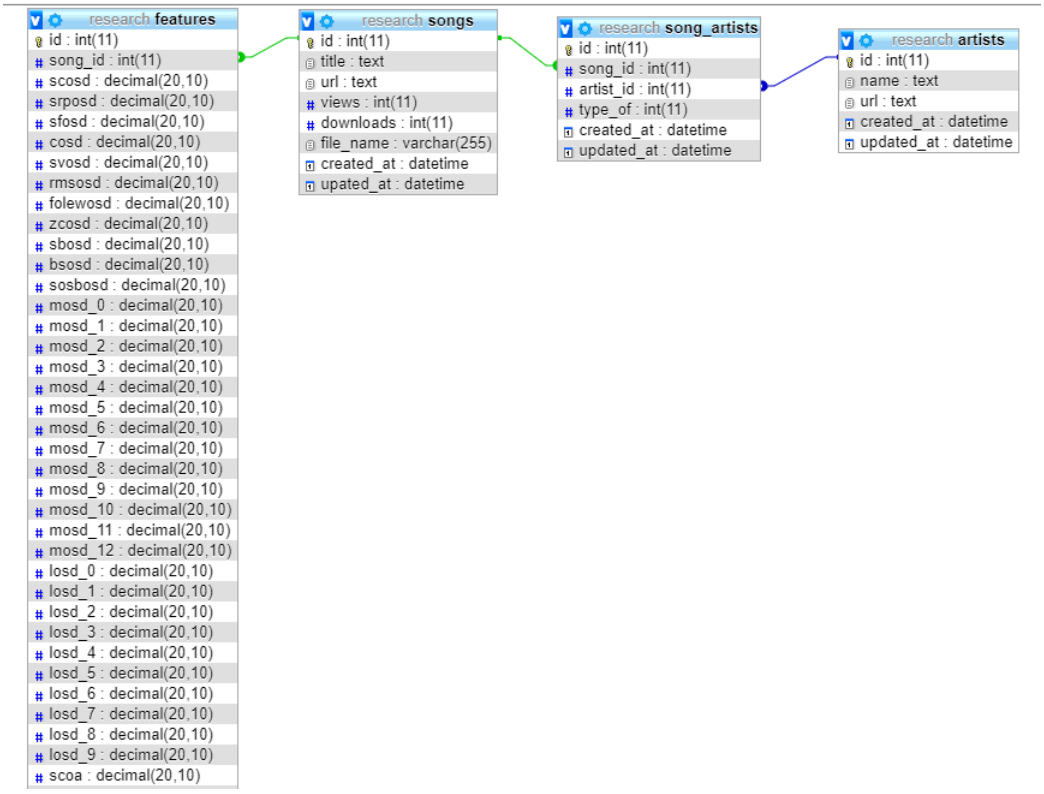

Figure 3.3: The Entity Relationship Diagram of the Database Design

3.3 Ethical Issues

Here the data collected was obtained only through publicly available channels (Online Websites). And since the data-set is only being used for non-commercial/academic purposes, it wouldn't violate any ethics.

[2]https://requests.readthedocs.io/en/master
[3]https://www.crummy.com/software/BeautifulSoup/bs4/doc

Chapter 4

Implementation

4.1 Web Scraping

T he web-scraping process was done according to a functional approach. First, the number of pages in the song listing website was determined by the page count indicator in each page, and there by the number of iterations were derived.

Once the loop iterates through each page, the song listing includes a link for each song, in order to view all the details related to that relevant song in another page. There the artist related data, and the download and views counts are indicated. These are obtained by extracting the content within the Hypertext Markup Language (HTML) elements. As soon as these details were available to the Python web-scraping script in a single fetch, they are stored in the relevant tables of the database. For each record inserted to a table, the database is configured to use an automatically incremented integer value name the **id** field, which is being used as the Primary Key of the table.

When these records are inserted for a single song details fetch, the song file is download by within the same HyperText Transfer Protocol (HTTP) session, by identifying the Uniform Resource Locator (URL) to the specific song. The content obtained when fetching the song file is written to the hard disk as a binary file. Once the song related details are inserted to the *songs* table in the database, the Primary Key (**id**) filed will be available to the script. For the purpose of further simplification, and in order to avoid duplicate file names being overwritten, this **id** field is used to rename the song file as `<id>.mp3` when saving to the hard disk.

4.2 Feature Extraction

The extraction was initially performed by the jAudio tool by compiling it as a Java ARchive (JAR) file. A database table was maintained to save these extracted features. The Python Librosa library was used to extract the duration, and tempo of each song and updated the same database table. The acronyms used for each

feature, is listed in the Table 4.1.

Attribute	**Acronym**
scosd	Spectral Centroid Overall Standard Deviation
srposd	Spectral Roll off Point Overall Standard Deviation
sfosd	Spectral Flux Overall Standard Deviation
cosd	Compactness Overall Standard Deviation
svosd	Spectral Variability Overall Standard Deviation
rmsosd	Root Mean Square Overall Standard Deviation
folewosd	Fraction of Low Energy Windows Overall Standard Deviation
zcosd	Zero Crossings Overall Standard Deviation
sbosd	Strongest Beat Overall Standard Deviation
bsosd	Beat Sum Overall Standard Deviation
sosbosd	Strength Of Strongest Beat Overall Standard Deviation
mosd (0 - 12)	MFCC Overall Standard Deviation
Losd (0-9)	LPC Overall Standard Deviation
scoa	Spectral Centroid Overall Average
srpoa	Spectral Rolloff Point Overall Average
sfoa	Spectral Flux Overall Average
coa	Compactness Overall Average
svoa	Spectral Variability Overall Average
rmsoa	Root Mean Square Overall Average
folewoa	Fraction Of Low Energy Windows Overall Average
zcoa	Zero Crossings Overall Average
sboa	Strongest Beat Overall Average
bsoa	Beat Sum Overall Average
sosboa	Strength Of Strongest Beat Overall Average
moa (0 - 12)	MFCC Overall Average
loa (0 - 9)	LPC Overall Average
tempo	Beats per Minute
duration	Song Length
popularity	Norm(downloads)*(Click Through Rate)

Table 4.1: The feature acronyms

The details of each of these features can be described as following.

Spectral Centroid Overall Standard Deviation

The center of mass of the power spectrum. This is the overall standard deviation over all windows.

Spectral Roll off Point Overall Standard Deviation

The fraction of bins in the power spectrum at which 85% of the power is at lower frequencies. This is a measure of the right-skewedness of the power spectrum. This is the overall standard deviation over all windows.

Spectral Flux Overall Standard Deviation

A measure of the amount of spectral change in a signal. Found by calculating the change in the magnitude spectrum from frame to frame. This is the overall standard deviation over all windows

Compactness Overall Standard Deviation

A measure of the noisiness of a signal. Found by comparing the components of a window's magnitude spectrum with the magnitude spectrum of its neighboring windows. This is the overall standard deviation over all windows

Spectral Variability Overall Standard Deviation

The standard deviation of the magnitude spectrum. This is a measure of the variance of a signal's magnitude spectrum. This is the overall standard deviation over all windows

Root Mean Square Overall Standard Deviation

A measure of the power of a signal. This is the overall standard deviation over all windows

Fraction of Low Energy Windows Overall Standard Deviation

The fraction of the last 100 windows that has an RMS less than the mean RMS in the last 100 windows. This can indicate how much of a signal is quite relative to the rest of the signal. This is the overall standard deviation over all windows

Zero Crossings Overall Standard Deviation

The number of times the waveform changed sign. An indication of frequency as well as noisiness. This is the overall standard deviation over all windows

Strongest Beat Overall Standard Deviation

The strongest beat in a signal, in beats per minute, found by finding the strongest bin in the beat histogram. This is the overall standard deviation over all windows

Beat Sum Overall Standard Deviation

The sum of all entries in the beat histogram. This is a good measure of the importance of regular beats in a signal. This is the overall standard deviation over all windows

Strength Of Strongest Beat Overall Standard Deviation

How strong the strongest beat in the beat histogram is compared to other potential beats. This is the overall standard deviation over all windows

MFCC Overall Standard Deviation

MFCC calculations based upon Orange Cow code This is the overall standard deviation over all windows. *This feature contains 13 parallel dimensions, which are named from mosd_0 to mosd_12.*

LPC Overall Standard Deviation

Linear Prediction Coefficients calculated using autocorrelation and Levinson-Durbin recursion. This is the overall standard deviation over all windows. *This feature contains 10 parallel dimensions, which are named from losd_0 to losd_9.*

Spectral Centroid Overall Average

The center of mass of the power spectrum. This is the overall average over all windows

Spectral Rolloff Point Overall Average

The fraction of bins in the power spectrum at which 85% of the power is at lower frequencies. This is a measure of the right-skewedness of the power spectrum. This is the overall average over all windows

Spectral Flux Overall Average

A measure of the amount of spectral change in a signal. Found by calculating the change in the magnitude spectrum from frame to frame. This is the overall average over all windows

Compactness Overall Average

A measure of the noisiness of a signal. Found by comparing the components of a window's magnitude spectrum with the magnitude spectrum of its neighboring windows. This is the overall average over all windows

Spectral Variability Overall Average

The standard deviation of the magnitude spectrum. This is a measure of the variance of a signal's magnitude spectrum. This is the overall average over all windows

Root Mean Square Overall Average

A measure of the power of a signal. This is the overall average over all windows

Fraction Of Low Energy Windows Overall Average

The fraction of the last 100 windows that has an RMS less than the mean RMS in the last 100 windows. This can indicate how much of a signal is quite relative to the rest of the signal. This is the overall average over all windows

Zero Crossings Overall Average

The number of times the waveform changed sign. An indication of frequency as well as noisiness. This is the overall average over all windows

Strongest Beat Overall Average

The strongest beat in a signal, in beats per minute, found by finding the strongest bin in the beat histogram. This is the overall average over all windows

Beat Sum Overall Average

The sum of all entries in the beat histogram. This is a good measure of the importance of regular beats in a signal. This is the overall average over all windows

Strength Of Strongest Beat Overall Average

How strong the strongest beat in the beat histogram is compared to other potential beats. This is the overall average over all windows

MFCC Overall Average

MFCC calculations based upon Orange Cow code This is the overall average over all windows. *This feature contains 13 parallel dimensions, which are named from moa_0 to moa_12.*

LPC Overall Average

Linear Prediction Coefficients calculated using autocorrelation and Levinson-Durbin recursion. This is the overall average over all windows. *This feature contains 10 parallel dimensions, which are named from loa_0 to loa_9.*

4.3 A Measure for the Popularity

The popularity score for each song is calculated according to the following formula. First the the download count for a single song is normalized to a value between 0 and 1 (inclusive) according to the following formula.

$Norm(download\_count) = \frac{download\_count\_of\_the\_song - min(downloads)}{max(downloads) - min(downloads)}$

Next the view count for a single song is normalized to a value between 0 and 1

(inclusive) according to the following formula.

$$Norm(view\_count) = \frac{view\_count\_of\_the\_song - min(views)}{max(views) - min(views)}$$

The Click Through Rate (Downloads per View) is obtained by the following formula.

$$click\_through\_rate = \frac{Norm(download\_count)}{Norm(view\_count)}$$

Finally the popularity score is derived by this formula.

$$The\_popularity\_score = Norm(download\_count) * click\_through\_rate$$

The click through rate is an optimal method in order to determine the popularity where the ranking happens based on the reach to a particular audience [19, 20]. A study by Zhou et al. presents their findings stating that the click through rate is an important factor in YouTube video recommendation which also concludes that the popularity of the video is based on the proper recommendation mechanism [21]. Therefore, here the click through rates is being used as the measure of the popularity score.

Experimentally by observing the distributions, the songs which have a download count less than or equal to 41,250 and a view count less than or equal to 93,874 are only considered. Other song statistics are considered as noise.

The popularity scores are a value between 0 and 1. The songs are divided into 3 classes based on these popularity values. The upper bound and the lower bound for each of these classes are decided by observing the elbow curve turning points when the ordered distribution of the popularity score values are plotted. The ***KneeLocator*** Python library is being used for this.

4.4 Model Training and Evaluation

The **eXtreme Gradient Boosting (XGBoost)** algorithm was used to train the data. Here an Ensemble learning is being used where multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Next the the impact of the features for the results are further indicated by the SHAP process.

Chapter 5

Results and Evaluation

5.1 The Dataset

The initial web scraped dataset consisted of 15,732 songs by 1,892 artists. 13,614 valid mp3 audio files were obtainable and after noise reductions and eliminating missing components in data, a set of 13,504 songs were used to extract features. This set was splitted as **70% for training and 30% for testing**. The reason to split in 7:3 ratio is, if more training data is available, a better model can be obtained, and if more testing data is available, a less variance in the results such as the accuracy and false positive rate can be expected. Therefore, a training set of around 9,452 songs were able to be used.

5.2 Song Popularity Scores Distribution

The popularity scores are normalized to a value between [0 – 1] based on the download counts and the view counts combined. In the Figure 5.1 a sorted representation (ascending order) of these scores are depicted for the visualization purpose.

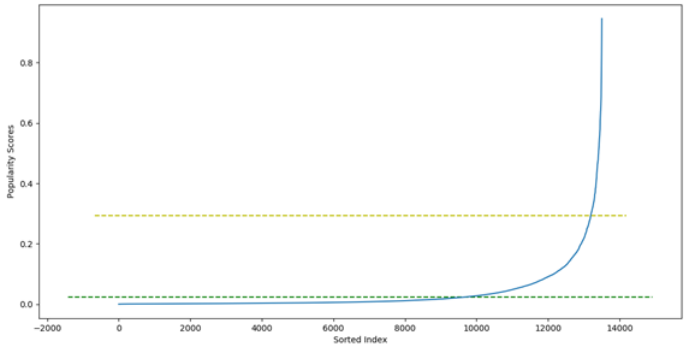

Figure 5.1: Song Popularity Scores Distribution

5.3 Song Popularity Classes Distribution

The popularity classes are determined based on the popularity scores according to the following conditions. The Figure 5.2 is the histogram obtained for the number of songs per each class. The Table 5.1 list outs the upper-bounds and the lower-bounds which were used when determining the classes.

Table 5.1: Determining the Song Popularity class based on the song popularity scores

Lower Bound	Upper Bound	Class
0	0.022996829493705295	Class 1
0.022996829493705295	0.294071579342863	Class 2
0.294071579342863	1	Class 3

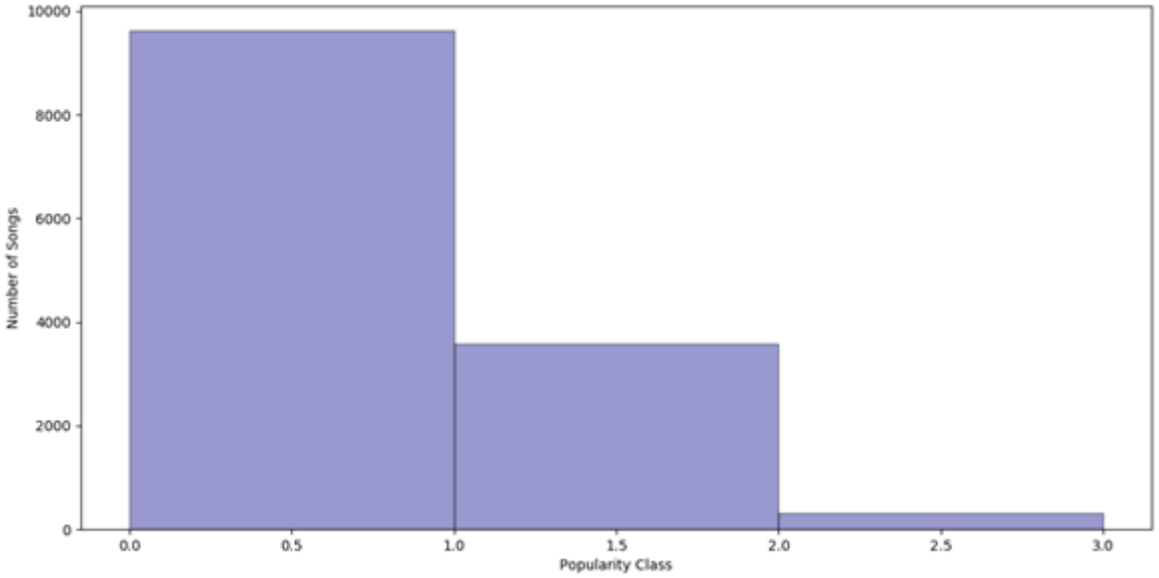

Figure 5.2: Song Popularity Classes Distribution

5.4 Evaluation

A classifier can be evaluated by the confusion matrix generated for a given sample. True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) were calculated for each test case. Then accuracy and FP rate was calculated for each test case. Reducing FP rate is significant as same as increasing the accuracy given that this method mainly focuses on identifying music on radio broadcasts. accuracy and FP rate can be calculated from the formulas given below.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{FP Rate} = \frac{FP}{FP + TN}$$

The Extended Gradient Boosting Classification Algorithm produced the following results.

$Shape\ \ of\ \ training\ \ data : (9452, 69)$
$Shape\ \ of\ \ testing\ \ data : (4052, 69)$
$accuracy\ \ score\ \ on\ \ train\ \ dataset : 0.7329665679221329$
$accuracy\ \ score\ \ on\ \ test\ \ dataset : 0.7129812438302073$

5.5 Explaining the Results

For further explaining the results the SHAP process was being used as depicted in the Figure 5.3. This produces the SHAP values for the significant features and their impact for the predictions [22].

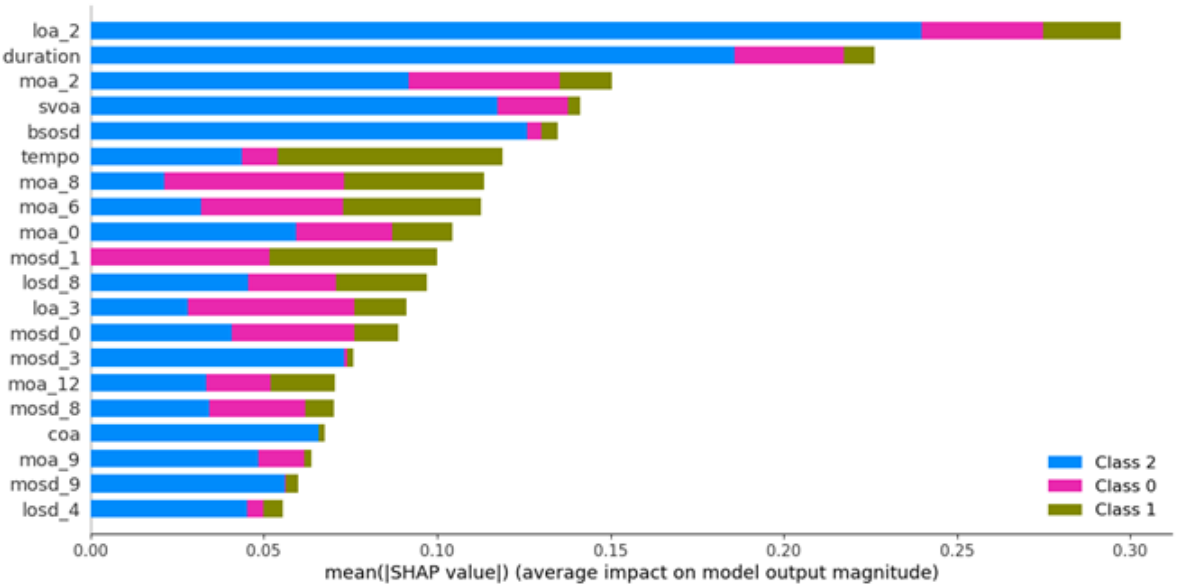

Figure 5.3: SHAP values of the trained model

As presented in the Figure 5.3 the feature **loa_2** (2^{nd} coefficient of the LPC overall average) indicates to be having the highest on the model output magnitude, which is then followed by the duration and the 2^{nd} coefficient of the MFCC overall average.

Chapter 6

Conclusions

6.1 Introduction

Here the review of the findings and how the research objectives were accomplished are described. While only the audio signal related features were used in this research as the key component for analysis, it yet produces a result that could be used for further analysis with different statistical approaches.

6.2 Conclusions About the Research Questions

The purpose of using machine learning algorithms in this project was to determine if such patterns exist in popular music and the experiments would suggest that the music audio signal consists of patterns which makes it a likable song for many. With the results produced by this research, it could also considered that the forms of repetition within a song impacts to the popularity, since it makes the song more memorable.

6.3 Conclusions About the Research Problems

The purpose of using machine learning algorithms in this project was to determine if such patterns exist in popular music and the experiments would suggest that the music audio signal consists of patterns which makes it a likable song for many. This research considered only the audio signal related data in order to predict whether a hit or not. The test results produced a 71% accuracy in predicting the correct popularity class for unseen data. And the audio frequency related features, duration and tempo would be key elements where a songs popularity depends on. As depicted in the SHAP values, the 'loa_2' (The second coefficient of the Linear Predictive Coding) has the most significance and contribution towards the output.

The Linear predictive coding is a method used mostly in audio signal processing and speech processing for representing the spectral envelope of a digital signal of

speech in compressed form, using the information of a linear predictive model. In addition to that, it is also used to identify the resonance [1] characteristics of audio [23]. This indicates that the repetition property which lies within the audio signal is a key element that correlates with the music popularity.

Additionally most of the similar researches in this area have been using the Million Song Dataset [4] where the features are already extracted using proprietary algorithms. Since this research describes how the features were extracted and the dataset is re-obtainable by anyone, it would be contributing to further research on predicting the popularity of songs.

6.4 Limitations

The webscraping was performed in the year 2019. So, the song download and view counts may differ when in a further webscrapping. And, the availability of the songs may subject to change in future depending on the decisions made by the website owners.

6.5 Implications for Further Research

This research only depends on the data collected at particular point of time. Since the trends of music change over time, a further research can be done by collecting a dataset over a period of time and analyze the results. Also, this research uses the XGBoost algorithm for processing. Other machine learning algorithms can be used to compare results in order to find the best model.

[1] The quality in a sound of being deep, full, and reverberating

References

[1] M. B. Holbrook and R. M. Schindler, "Some exploratory findings on the development of musical tastes," vol. 16, no. 1, p. 119.

[2] M. Nasreldin, "Song popularity predictor," May 2018.

[3] R. Dhanaraj and B. Logan, "Automatic prediction of hit songs.," in *ISMIR*, pp. 488–491, 2005.

[4] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," 2011.

[5] J. Q. Pham, "Predicting song popularity," 2015.

[6] F. Pachet and P. Roy, "Hit song science is not yet a science.," in *ISMIR*, pp. 355–360, 2008.

[7] Y. Ni, R. Santos-Rodriguez, M. Mcvicar, and T. De Bie, "Hit song science once again a science," in *4th International Workshop on Machine Learning and Music: Learning from Musical Structure, Sierra Nevada, Spain*, Citeseer, 2011.

[8] N. Borg and G. Hokkanen, "What makes for a hit pop song? what makes for a pop song," *Unpublished thesis, Stanford University, California, USA*, 2011.

[9] J. Fan and M. Casey, "Study of chinese and uk hit songs prediction," in *Proceedings of International Symposium on Computer Music Multidisciplinary Research*, pp. 640–652, 2013.

[10] D. Herremans, D. Martens, and K. Sörensen, "Dance hit song prediction," *Journal of New Music Research*, vol. 43, no. 3, pp. 291–302, 2014.

[11] T. Paranagama and A. Ariyaratne, "A machine learning approach to classify sinhala songs based on user ratings," 08 2017.

[12] B. Shao, D. Wang, T. Li, and M. Ogihara, "Music recommendation based on acoustic features and user access patterns," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 8, pp. 1602–1611, 2009.

[13] M. Airoldi, D. Beraldo, and A. Gandini, "Follow the algorithm: An exploratory investigation of music on youtube," *Poetics*, vol. 57, pp. 1–13, 2016.

[14] S. Zannettou, M. Sirivianos, J. Blackburn, and N. Kourtellis, "The web of false information: Rumors, fake news, hoaxes, clickbait, and various other shenanigans," *Journal of Data and Information Quality (JDIQ)*, vol. 11, no. 3, pp. 1–37, 2019.

[15] H.-Y. Chang, S.-C. Huang, and J.-H. Wu, "A personalized music recommendation system based on electroencephalography feedback," *Multimedia Tools and Applications*, vol. 76, no. 19, pp. 19523–19542, 2017.

[16] H. MacGillivray, J. M. Utts, and R. F. Heckard, *Mind on statistics*. Cengage Learning, 2014.

[17] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[18] C. McKay, I. Fujinaga, and P. Depalle, "jaudio: A feature extraction library," in *Proceedings of the International Conference on Music Information Retrieval*, pp. 600–3, 2005.

[19] P. Chandar and B. Carterette, "Estimating clickthrough bias in the cascade model," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 1587–1590, 2018.

[20] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, *et al.*, "The youtube video recommendation system," in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 293–296, 2010.

[21] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 404–410, 2010.

[22] S. Rathi, "Generating counterfactual and contrastive explanations using shap," *arXiv preprint arXiv:1906.09293*, 2019.

[23] N. Dave, "Feature extraction methods lpc, plp and mfcc in speech recognition," *International journal for advance research in engineering and technology*, vol. 1, no. 6, pp. 1–4, 2013.

Appendices

Appendix A

Code Listings

A.1 The Python Script for Webscraping

```
import requests
import urllib.request
import time
from bs4 import BeautifulSoup
from datetime import datetime


import re

import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  passwd="",
  database="songs"
)

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108
Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
    'DNT': '1',
    'Host': 'sarigama.lk',
    'Range': 'bytes=0-',
    'Sec-Fetch-Mode': 'no-cors',
    'Sec-Fetch-Site': 'same-origin'
}


def addArtist(name, url):
```

```
    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM `sarigama_artists` WHERE `url` =
'"+url+"' "

    selectCursor.execute(search_sql)

    isExist = selectCursor.fetchall()

    if(len(isExist) > 0):
        return isExist[0][0]
    else:
        insertCursor = mydb.cursor()

        insert_sql = "INSERT INTO `sarigama_artists` (`name`, `url`,
`created_at`) VALUES(%s, %s, %s)"

        vals = (name, url, str(datetime.now()))

        insertCursor.execute(insert_sql, vals)
        mydb.commit()

        artist_id = insertCursor.lastrowid

        return artist_id

def addSong(title, url, views, downloads):
    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM `sarigama_lk` WHERE `url` =
'"+url+"' "

    selectCursor.execute(search_sql)

    isExist = selectCursor.fetchall()

    if(len(isExist) > 0):
        return isExist[0][0]
    else:
        insertCursor = mydb.cursor()

        insert_sql = "INSERT INTO `sarigama_lk` (`title`, `url`,
`views`, `downloads`, `created_at`) VALUES(%s, %s, %s, %s, %s)"

        vals = (title, url, str(views), str(downloads),
str(datetime.now()))
```

```
        insertCursor.execute(insert_sql, vals)
        mydb.commit()

        song_id = insertCursor.lastrowid

        return song_id


def addSongArtist(song_id, artist_id, type_of):
    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM ‘sarigama_song_artists‘
WHERE ‘song_id‘ = ’"+str(song_id)+"’ AND ‘artist_id‘ =
’"+str(artist_id)+"’ AND ‘type_of‘ = ’"+str(type_of)+"’"

    selectCursor.execute(search_sql)

    isExist = selectCursor.fetchall()

    if(len(isExist) > 0):
        return isExist[0][0]
    else:
        insertCursor = mydb.cursor()

        insert_sql = "INSERT INTO ‘sarigama_song_artists‘
(‘song_id‘, ‘artist_id‘, ‘type_of‘,‘created_at‘) VALUES(%s, %s,
%s, %s)"

        vals = (str(song_id), str(artist_id), str(type_of),
str(datetime.now()))

        insertCursor.execute(insert_sql, vals)
        mydb.commit()

        song_artist_id = insertCursor.lastrowid

        return song_artist_id

def updateSong(song_id, file_name):
    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM ‘sarigama_lk‘ WHERE ‘id‘ =
’"+str(song_id)+"’ "

    selectCursor.execute(search_sql)

    isExist = selectCursor.fetchall()
```

```
    if(len(isExist) > 0):
        updateCursor = mydb.cursor()

        update_sql = "UPDATE `sarigama_lk` SET `file_name` =
'"+file_name+"' WHERE `id` = '"+str(song_id)+"'"



        updateCursor.execute(update_sql)
        mydb.commit()

        row_count = updateCursor.rowcount


        return row_count
    else:
        return isExist[0][0]



def downloadSong(song_id, header_name, force_song_file_url, s):
    song = s.get(force_song_file_url, headers=headers)

    if(str(song.status_code)[0] != '2'):
        return -1

    online_file_name = song.headers.get(header_name).replace('"',
'')

    file_name = str(song_id)


    regex = re.compile('[^\s0-9a-zA-Z\_\-\.]')

    file_name = regex.sub('', file_name)+online_file_name[-4:]

    open("./songs/"+file_name, "wb").write(song.content)

    return updateSong(song_id, file_name)




def fetchSong(song_url):
    print("\n\n=======\nSong url: ", song_url, "\n=======\n")
    s = requests.Session()

    song_resp = s.get(song_url)
```

```
    song_soup = BeautifulSoup(song_resp.text, "html.parser")


    meta_divs = song_soup.findAll('div', {'class': 'item-action
m-b'})

    if(len(meta_divs) > 0):
        counts = meta_divs[0].findAll('span')
        downloads = int(counts[0].text)
        views = int(counts[1].text)

        song_title_h1 = song_soup.find('h1')
        song_title = song_title_h1.text

        print("song title: ", song_title)


        song_id = addSong(song_title, song_url, views, downloads)

        print("\n----------\n Views: ", views, " , downloads: ",
downloads, "\n----------\n" )

        artists_table = meta_divs[1].find('table')
        artists_table_trs = artists_table.findAll('tr')

        artists = {'singers':[], 'lyricist':[], 'musicians':[]}

        for atr in range(len(artists_table_trs)):

            icon = artists_table_trs[atr].findChildren('i',
{'class':'icon'})
            artist_anchors = artists_table_trs[atr].findAll('a')

            if("singer" in " ".join(icon[0]['class'])):
                for si in range(len(artist_anchors)):
                    artist_name = artist_anchors[si].text.strip()
                    artist_url = artist_anchors[si]['href']
                    artists['singers'] += [{'name': artist_name,
'url': artist_url }]

                    artist_id = addArtist(artist_name, artist_url)
                    song_artist_id = addSongArtist(song_id,
artist_id, 1)

            if("lyrics" in " ".join(icon[0]['class'])):
                for si in range(len(artist_anchors)):
                    artist_name = artist_anchors[si].text.strip()
                    artist_url = artist_anchors[si]['href']
```

```
                    artists['lyricist'] += [{'name': artist_name,
'url': artist_url }]

                    artist_id = addArtist(artist_name, artist_url)
                    song_artist_id = addSongArtist(song_id,
artist_id, 2)

            if("music" in " ".join(icon[0]['class'])):
                for si in range(len(artist_anchors)):
                    artist_name = artist_anchors[si].text.strip()
                    artist_url = artist_anchors[si]['href']
                    artists['musicians'] += [{'name': artist_name,
'url': artist_url }]

                    artist_id = addArtist(artist_name, artist_url)
                    song_artist_id = addSongArtist(song_id,
artist_id, 3)

        print("---------artists:\n", artists, "\n------------\n")



    download_url_div = song_soup.find('div',
{'class':'download-play-btn-group'})
    if(download_url_div is not None and len(download_url_div) > 0):

        download_btns = download_url_div.findAll('a')
        if(download_btns is not None and len(download_btns) > 1):

            song_file_url = download_btns[1]['href']
            song_file_page = s.get(song_file_url)
            song_file_soup = BeautifulSoup(song_file_page.text,
"html.parser")

            divs = song_file_soup.find('div', {'class':'col-md-12
text-center'})
            links = divs.findAll('a')

            if(links is not None and len(links) > 0):
                download_status = downloadSong(song_id,
'Content-Disposition', links[0]['href'], s)
                if(download_status == -1):
                    audio_src = song_soup.findAll("source",
{"itemprop":"audio"})
                    mp3_link = audio_src[0]['src']
                    download_status = downloadSong(song_id,
'filename', mp3_link, s)
```

```
            else:
                print("\n+=+=+=+=+=+=+=+=+=+=+\nNo links available to
download\n+=+=+=+=+=+=+=+=+=+=+\n")


        else:
            print("\n+=+=+=+=+=+=+=+=+=+=+\nNo button available to
download\n+=+=+=+=+=+=+=+=+=+=+\n")
    else:
        print("\n+=+=+=+=+=+=+=+=+=+=+\nNo button divs available to
download\n+=+=+=+=+=+=+=+=+=+=+\n")


def fetchPage(base_url, page_no):
    page_url = base_url + "&page="+str(page_no)
    page_resp = requests.get(page_url)
    page_soup = BeautifulSoup(page_resp.text, "html.parser")

    song_divs = page_soup.findAll('div', {"class": "item-media
song"})

    song_divs_count = len(song_divs)

    print("\n==============\n\n Fetching page:
"+str(page_no)+"......\n\n Found ",song_divs_count," songs in
page:",page_no,", starting with:",base_url[-1],"\n==============\n")

    for s in range(song_divs_count):
        song_links = song_divs[s].findAll('a')
        for t in range(len(song_links)):
            if(song_links[t]["href"].lower().startswith("https")):
                fetchSong(song_links[t]["href"])
                break



def fetchPages(base_url, pg_limit, current_page):
    for x in range(current_page, pg_limit):
        fetchPage(base_url, x)



for i in range(86,91):
   c = chr(i)
   print("\n============================\nSongs starting with
",c,"\n============================\n")
   pg_url = "https://sarigama.lk/songs?starts-with="+c
```

```
    pg_resp = requests.get(pg_url)
    pg_soup = BeautifulSoup(pg_resp.text, "html.parser")

    print("\n====================\nFetching page 1 for song listing
starting with: ",c,"\n====================\n")

    if(i == 86):
        first_page = 2
    else:
        first_page= 1

    fetchPage(pg_url, first_page)

    pagination = pg_soup.findAll('a', {"class": "page-link"})
    if(len(pagination) > 2):
        last_pg = int(pagination[len(pagination) - 2].text)
        print("\n==============\nFound upto",(last_pg + 1)," pages.
Fetching each.\n==============\n")
        fetchPages(pg_url, (last_pg + 1), (first_page + 1))
```

A.2 Feature Extraction

```
import random
import csv
import time
from decimal import Decimal
import math

from kneed import KneeLocator
import matplotlib.pyplot as plt

import seaborn as sns

import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  passwd="",
  database="songs"
)


song_ids = []
```

```
def getSong(song_id):
    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM ‘sarigama_lk‘ WHERE ‘id‘ =
’"+str(song_id)+"’ AND file_name IS NOT NULL "

    selectCursor.execute(search_sql)

    song_data = selectCursor.fetchone()

    return list(song_data)

def isFileExist(song_id):
    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM ‘sarigama_lk‘ WHERE ‘id‘ =
’"+str(song_id)+"’ AND file_name IS NOT NULL "

    selectCursor.execute(search_sql)

    isExist = selectCursor.fetchall()

    if(len(isExist) > 0):
        return True
    else:
        return False


def getAll(limit = -1, offset = 0):

    ids = "’,’".join(list(map(str, song_ids)))

    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM ‘sarigama_features‘ WHERE ‘song_id‘
NOT IN (’"+ids+"’) AND ‘downloads‘ <= 41250 AND ‘views‘ <= 93874 "
    if(limit > 0):
        search_sql += " LIMIT "+str(limit)+" OFFSET "+str(offset)

    selectCursor.execute(search_sql)

    result = selectCursor.fetchall()

    for x in result:
        song_ids.append(x[0])
```

```
def getFeatureAttr():
    selectCursor = mydb.cursor()
    search_sql = "SELECT * FROM ‘sarigama_temp‘ WHERE ‘downloads‘ <=
41250 AND ‘views‘ <= 93874 "

    selectCursor.execute(search_sql)

    feature_attrs = []

    for i in selectCursor.description:
        feature_attrs.append(i[0])
    return feature_attrs




def getRandomFeatures(limit = -1, offset = 0):

    ids = "','".join(list(map(str, song_ids)))

    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM ‘sarigama_temp‘ WHERE ‘song_id‘ NOT
IN ('"+ids+"') AND ‘downloads‘ <= 2500 AND ‘views‘ <= 50000 ORDER BY
RAND() "
    if(limit > 0):
        search_sql += " LIMIT "+str(limit)+" OFFSET "+str(offset)

    selectCursor.execute(search_sql)

    result = selectCursor.fetchall()

    song_features = []

    for x in result:
        feature = []
        for y in x:
            if(isinstance(y, Decimal)):
                feature.append(float(y))
            else:
                feature.append(y)
        song_ids.append(x[0])
        song_features.append(feature)

    return song_features


def getAllFeatures(limit = -1, offset = 0):
```

```
    ids = "','".join(list(map(str, song_ids)))

    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM `sarigama_temp` WHERE `song_id` NOT
IN ('"+ids+"') AND `downloads` <= 41250 AND `views` <= 93874 "
    if(limit > 0):
        search_sql += " LIMIT "+str(limit)+" OFFSET "+str(offset)

    selectCursor.execute(search_sql)

    result = selectCursor.fetchall()

    song_features = []

    for x in result:
        feature = []
        for y in x:
            if(isinstance(y, Decimal)):
                feature.append(float(y))
            else:
                feature.append(y)
        song_ids.append(x[0])
        song_features.append(feature)

    return song_features


def getRandom(limit = 300):

    ids = "','".join(list(map(str, song_ids)))

    selectCursor = mydb.cursor()

    search_sql = "SELECT * FROM `sarigama_lk`  WHERE file_name
IS NOT NULL AND `id` NOT IN ('"+ids+"') ORDER BY RAND() LIMIT
"+str(limit)

    selectCursor.execute(search_sql)

    result = selectCursor.fetchall()

    for x in result:
        song_ids.append(x[0])

def writeToCSV(fieldnames, data, prefix = "features_", dropping=[]):
```

```
    dropped_indexes = []
    headers = []

    for a in range(len(dropping)):
        if(dropping[a] in fieldnames):
            dropping_index = fieldnames.index(dropping[a])
            dropped_indexes.append(dropping_index)

    for b in range(len(fieldnames)):
        if(b not in dropped_indexes):
            headers.append(fieldnames[b])


    filename = "./datasets/"+prefix+str(time.time())+".csv"
    with open(filename, 'w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=headers)
        writer.writeheader()

        for d in range(len(data)):
            vals = {}
            datum = data[d]
            for f in range(len(datum)):
                if(f in dropped_indexes):
                    continue
                else:
                    vals[fieldnames[f]] = datum[f]

            writer.writerow(vals)

    print("CSV file", filename, "created!")




song_features = getAllFeatures()
num_features = len(song_features)


downloads = []
views = []

for x in range(num_features):
    song_feature = song_features[x]
    downloads.append(song_feature[-4]) ## the number of downloads
    views.append(song_feature[-5])


min_d = min(downloads)
```

```
max_d = max(downloads)

min_v = min(views)
max_v = max(views)

d_norms = []
v_norms = []

popularities = []
popularity_scores = []

for i in range(num_features):
    song_feature = song_features[i]
    download_count = song_features[i][-4]
    view_count = song_features[i][-5]
    ctr = download_count / view_count
    d_norm = (download_count - min_d)/(max_d - min_d)
    v_norm = (view_count - min_v)/(max_v - min_v)


    popularity = d_norm * ctr
    is_popular = 0
    if(popularity >= 0.294071579342863):
        is_popular = 2
    elif(popularity >= 0.022996829493705295):
        is_popular = 1

    song_features[i][-1] =  is_popular
    popularities.append(is_popular)

    d_norms.append(d_norm)
    v_norms.append(v_norm)


    popularity_scores.append(popularity)

dataset = song_features



random.shuffle(dataset)
dt_len = len(dataset)
train_len = int(dt_len*0.7)
test_len = dt_len - train_len
train_data = dataset[:train_len]
test_data = dataset[test_len*-1:]

feature_attr = getFeatureAttr()
```

```
yt = popularity_scores


yt.sort()
yt = yt[::1]
x = [i for i in range(len(yt))]
kn = KneeLocator(x, yt, curve='convex', direction='increasing',
online=False)
print("Knee:", kn.knee, " - ", yt[kn.knee] )
print("Elbow:", kn.elbow, " - ", yt[kn.elbow] )

plt.plot(x, yt)

plt.hlines(yt[kn.knee], plt.xlim()[0], plt.xlim()[1], colors='y',
linestyles='dashed')
plt.hlines(0.022996829493705295, plt.xlim()[0], plt.xlim()[1],
colors='g', linestyles='dashed')

plt.show()


sns.distplot(popularities, hist=True, kde=False, bins=[0,
1, 2, 3], color='darkblue', hist_kws={'edgecolor':'black'},
kde_kws={'linewidth': 4})
plt.show()
```

A.3 Processing using XGBoost

```
import pandas as pd
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

import xgboost as xgb
import matplotlib.pyplot as plt

import shap

features_data = pd.read_csv('./datasets/features_1579111035.0951855.csv')
train_data, test_data = train_test_split(features_data, test_size =
0.3)
```

```
# shape of the dataset
print('Shape of training data :',train_data.shape)
print('Shape of testing data :',test_data.shape)

# Now, we need to predict the missing target variable in the test
data
# target variable - Survived

# seperate the independent and target variable on training data
train_x = train_data.drop(columns=['popularity'],axis=1)
train_y = train_data['popularity']

# seperate the independent and target variable on testing data
test_x = test_data.drop(columns=['popularity'],axis=1)
test_y = test_data['popularity']

model = XGBClassifier()

# fit the model with the training data
model.fit(train_x,train_y)


# predict the target on the train dataset
predict_train = model.predict(train_x)
print('\nTarget on train data',predict_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(train_y,predict_train)
print('\naccuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_test = model.predict(test_x)
print('\nTarget on test data',predict_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(test_y,predict_test)
print('\naccuracy_score on test dataset : ', accuracy_test)


shap_values = shap.TreeExplainer(model).shap_values(train_x)
shap.summary_plot(shap_values, train_x, plot_type="bar")
```