Haaga-Helia
ammattikorkeakoulu Oy

Hit song classification with spotify audio features of Billboard list songs

Juuso Lindholm

Bachelor's Thesis
Information theory
2022

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

xx.xx.2022

Abstract

Author
Juuso Lindholm

Degree
Bachelor of Business Administration

Report/thesis title

Number of pages and appendix pages

In this work an experiment was made to predict hit and non-hit songs based on audio features provided by Spotify. Four different models were compared with the dataset.
The billboard data was first collected from Kaggle. The collected data did not include a Spotify id connecting the song from Kaggle to a song in Spotify collections. In order to retrieve the audio features of a song, that will be used as input to the models, a mapping had to be implemented. The implementation was done by retrieving songs from Spotify search API and matching the name of song and artist to the ones used to make the query. During the matching process, some words in the songs were excluded that were referring to different versions of the original song.
After retrieving the track information, the audio features were collected from Spotify audio features API.
After storing the hit song features, the non-hit songs were collected by using the Spotify album id in the song information. The non-hit songs were randomly sampled from the album and a check to verify the song not being the same as what was used to make the query. Finally, the audio features for the non-hit songs were collected the same way as for the hit songs.
The data was processed, transformed and hyperparameters were searched.

Keywords
Hit song science, machine learning

Index

1 Introduction

Throughout history music has been a way to influence people. It has an affect on humans, sometimes also other living beings, from calming and soothing to energizing and maddening. Sounds resonate large groups of people differently in a spectrum of different emotions. Many can say of experiencing that feeling when a song sounds good. Why it does, is hard to explain even by the individual having the feeling.

Humans and other beings interpret sound differently than machines and conversion mechanisms to store audio in digital form for machines have been developing a lot from the early 70s (Wikipedia, 2021). With the digitalization of audio, came the research field of music information retrieval, MIR (ISMIR, 2021; MIRAX, 2021; David Moffat, David Ronan, Joshua D. Reiss, 2015).

For MIR, a developer platform called The Echo Nest was created (The Echo Nest blog 2021). The Echo Nest was later sold to Spotify (The Echo Nest blog, 6.3.2014; Darrell Etherington, 6.3.2014) and the audio features are retrievable from their API for developers (Spotify API documentation, 2021). The audio features try to represent the song with just a few features, instead of the full spectrum of the digitally converted sound data.

The purpose of this work is to compare different models that are trained with the Spotify provided audio feature data.

This is not a novel attempt and the whole field of hit song science has been criticized by ISMR (François Pachet & Pierre Roy, 2008), but a few publications have later reported good results (Dorien Herremans, David Martens & Kenneth Sörensen 2014; Kai Middlebrook & Kian Sheik 2019).

Songs are usually charted by multiple institutions based on metrics usually referring to sales and streaming performances (Wikipedia, 2021). For example, Billboard magazine, a well-known brand in music chart domain, bases its Top 100 chart on measurable radio and online streaming and sales metrics in United States (Billboard 2018).

In (Kai Middlebrook & Kian Sheik 2019) Billboard top 100 chart was used to define a hit song.

The studies done before, chose random sampled songs to act as non-hit songs (Kai Middlebrook & Kian Sheik 2019). Here, instead of random sampling, the non-hit songs are collected randomly from the albums related to hit songs. Intuition is that the audio features of the non-hits would be closer to the hit songs as the sound sphere of songs from same artist and album is in general closer than the content of different artists.

As labels are used to train machine learning models, the task will be in machine learning jargon, a supervised learning task (Aurélien Géron 2017, Types of Machine Learning Systems, Supervised/Unsupervised Learning).

2 Theoretical framework

(François Pachet & Pierre Roy, 2008) came into a conclusion that “the popularity of a song cannot be learnt by using state-of-the-art machine learning techniques with two sets of reasonable audio features.” while they were trying to validate claim made by (Ruth Dhanaraj & Beth Logan 2005) of finding a way to map features to a hit song. The conclusions were drawn using a comprehensive set of three datasets with different and distinguishable feature sets. The total amount of data was 32,000 songs. This was a lot bigger than 1700 song dataset used in (Ruth Dhanaraj & Beth Logan 2005). (Aurélien Géron 2017, Main Challenges of Machine Learning) lists insufficient quantity of training data as being an issue when practicing machine learning. Increasing the amount of data like in (François Pachet & Pierre Roy, 2008) versus (Ruth Dhanaraj & Beth Logan 2005) and the results being unimpressive, would it suggest that the dataset doesn’t have enough representative cases. This is called sampling noise and it will render a badly generalizable model, meaning it will perform badly with real world data.
However, in (Meghan Neal, 2015) article writes about a study (Dorien Herremans, David Martens & Kenneth Sörensen 2014) that does a decent job of predicting hit dance songs of 2015. They used The Echo Nest API to collect features for 697 unique Official Charts Company hit listings and 2755 unique Billboard hit listings, between the years 1985-2013. Only the audio features were used for the training of the model, concluding that the popularity of dance songs can be learnt from the audio features provided by the Echo Nest. Later studies using the Echo Nest audio features have given (Minna Reiman & Philippa Örnell, 2018; E. Georgieva, Marcella Suta, N. Burton, 2018; Kai Middlebrook & Kian Sheik 2019; Adewale Adeagbo, 2020) more mixed results.
A conclusion can be drawn from the work before that the audio features pulled from the Echo Nest can be used to possibly predict hit songs with a decent accuracy in specified song domains, because of this and the availability of relevant data, the Echo Nest audio features will be used in this study too.

2.1 Hit song definition

When predicting a hit song, the hit song should be defined. (Dorien Herremans, David Martens & Kenneth Sörensen 2014) used singles dance archive from the Official Charts Company and Billboard. Later (Dorien Herremans & Tom Bergmans 2017) in a follow up study to focus on the impact of early adaptors in hit song predictions, “The Ultratop 50” listing was used to define a hit song.

The “Billboard hit 100” listing is used in more recent studies (E. Georgieva, Marcella Suta & N. Burton, 2018; Minna Reiman & Philippa Örnell, 2018; Kai Middlebrook & Kian Sheik 2019) where the song “genre” was not more specifically scoped.
As the scope of the song genre is not taken into consideration in this research, the “Billboard Hot 100” listing will be used to define a hit song.

As the songs are labelled as a hit or non-hit song, based on if they are found on the Billboard Hot 100 list and these labels are used to teach the model, the task itself can be labelled as a supervised learning task. More specifically, because the song can either be a hit or a non-hit, it is a binary classification task.
In machine learning classifier model is usually used for a classification task, although it is not uncommon to see a regression model (Aurélien Géron 2017, p. 103) to be used in a binary classification like logistic regression in (Dorien Herremans, David Martens & Kenneth Sörensen 2014; Dorien Herremans & Tom Bergmans 2017; Minna Reiman & Philippa Örnell 2018; E. Georgieva, Marcella Suta & N. Burton, 2018; Kai Middlebrook & Kian Sheik 2019) by defining a threshold for the output value. If the predicted value is over the threshold it is said to belong in class 1 and if lower in class 2.
Also support vector machines (SVM) and neural networks (NN) have been a popular choice for the studies before.
In (Kai Middlebrook & Kian Sheik 2019) random forest rendered impressive results.
For the empirical section of this work, a logistic regression, SVM, NN and random forest model is used via scikit-learn API (Scikit-learn API documentation, 2021) implementations.

2.2 Evolution of pop music

The sound of music has changed during the years. When hearing the sound of the 80s or 90s a distinct feature of the sound can be distinguished in pop music even for the non-musical. (Matthias Mauch, Robert M. MacCallum, Mark Levy, Armand M. Leroi 2015) used audio analysis on the billboard hot 100 list songs from 1960s to 2010s to harvest audio features they called “topics” and concluded that the topics did vary among different time periods.

2.3 Datasets & features overview

This section will define the source of the data used in this work and gives an overview of the data available from these sources.

In a machine learning problem, the data is the most important factor. No matter how marvellous the machine learning model itself is, badly processed data will alter radically the results. There are a couple of well-known weak spots that should always be considered when cleaning and sorting the data into different sets for the model's consumption. These processing problems are addressed as well in this section.

2.4 Source data

The data is collected from two different domains.
Kaggle is providing the information about songs that are featured in a Billboard list.
The musical features that define the sound of the song itself and are used to train the model are collected from Spotify.

2.4.1 Kaggle

For the source data Kaggle Billboard "The Hot 100" Songs – dataset is used (Kaggle dataset, 2021). The songs are from lists between 1958-2021 and it contains 24620 unique songs in total 328 487 list song records.
The information about the songs from this source that are used in the empirical part are:

date;	date of the song being on the list.
song;	name of the song.
artist;	name of the song's artist.

The data is mentioned to be collected from Billboard on their web-site listing.
In this work all unique songs in this dataset are considered to be hit songs.

2.4.2 Spotify

Spotify provides an extensive collection of data related to the songs they have on their collections. This API is at the time of this writing available for anyone with a Spotify account.
All songs are not available for all geographic locations, nor is every single song that is featured in the Kaggle Billboard dataset in the collections of Spotify. However, the number of songs unavailable is so low that it should not have a significant impact on the study done here.
The Spotify audio_features API provided features used for training the models in this work are the following: **acousticness**, **danceability**, **duration_ms**, **energy**, **instrumental-**

ness, **key**, **liveness**, **loudness**, **mode**, **speechiness**, **tempo**, **time_signature** and **valence** (Spotify API documentation, 2021 Get track's audio features). The features are described in table 1.

acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
duration_ms	The duration of the track in milliseconds.
energy	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
instrumentalness	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
key	The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the

	quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
time_signature	An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4".
valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Table 1: Audio feature descriptions

Also **album_id:**s from the Kaggle billboard dataset songs are used to query the Spotify album API for the album information of the album in which the song was released. The album data is collected to get the **Spotify song_id**:s from the same album.
This step is done in order to introduce “non-hit” songs in the total dataset.
The logic for this is that a song that is released in the same album than the song which was featured in the Billboard hit list acts as a good comparison when trying to find the

border between a Billboard hit list and non-hit list. Most of the songs in the same album are from the same artist so the style and sound should be close to each other, but only one of them had the features in musical domain which made it to be appended in the Billboards hit list collection.
More specified information about the features and different API endpoints in (Spotify API documentation, 2021).

2.5 Data quality

The quality of the data can have a huge impact on the results and as mentioned in (Aurélien Géron 2017, p. 25) most data scientist will spend most of their time cleaning up the data. Also (Aurélien Géron 2017, p. 25)) mentions that usually clear outliers are in most cases just discarded in the resulting dataset and that another general case is that some data is missing.
These cleaning operations are addressed in this work and are mentioned in the above section about the source data, by discarding songs from which no audio features are available from the final dataset.
“With few exceptions, Machine Learning algorithms don’t perform well when the input numerical attributes have very different scales” (Aurélien Géron 2017, p.66)
(Aurélien Géron 2017, p.66) lists feature scaling as one of the most important transformations of the data in machine learning setting and lists normalization (MinMax scaling) and standardization as the two most common ways to apply feature scaling.
In normalization the feature values are fitted between 0 and 1 based on the original values. This method is more vulnerable to outliers as one high value will effect on all resulting values.
Standardization is less affected by outliers, but it does not bound the output values between 0s and 1s, which is a problem for some ML methods.
In this work some of the audio features are already in a scaled form and they will not be transformed in during the cleaning.
However, all the audio features that are not in scale of 0-1 will be scaled to be between 0-1 with (Scikit-learn API 2021, Minmax scaler) and in a separate dataset with (Scikit-learn API 2021, StandardScaler), these include **duration_ms**, **loudness** and **tempo**.
In (Aurélien Géron 2017, p. 64) it is showed by example that categorical data could be just changed into a numerical form where a number represents a category, but it is pointed out that “ML algorithms will assume that two nearby values are more similar than two distant values.” and to even out the closeness relation, usually onehot-encoding is used. A onehot-encoder will create a list where every category is either 1 or 0 depending on if the data instance belongs to it.

In this work categorical features like **timeSignature**, **key** and **mode** will be one hot encoded with (Scikit-learn API 2021, OneHotEncoder).
From the album release dates, the years will be parsed into an additional time feature. This time feature will be the end of the year. For example, 2021 would yield a time feature 21. The time features will be scaled between 0 and 1.
Technically the time features could be also onehot-encoded, but it is decided not to as years closer to each other can be seen as being more similar, when thinking about the sound in a time period.

2.6 Balancing & sampling

As put in (Aurélien Géron 2017, p. 24) “It is crucial to use a training set that is representative of the cases you want to generalize to. If the sample is too small, you will have sampling noise, but even very large samples can be nonrepresentative if the sampling method is flawed. This is called sampling bias “
Sampling bias is a tricky problem to address. In (Kai Middlebrook & Kian Sheik 2019) the not-hit songs were totally randomly sampled from a mentioned song database. The good accuracy performance could be explained by this sample being in its musical features so far off from the hit song samples that finding the defining line of hits and non-hits could be easier.
As the non-hits in this work should be by nature closer in the feature domain of the hit songs, the space between hits and non-hits can be theorized to be less wide.
(Kai Middlebrook & Kian Sheik 2019) also mentions the same problem which is that there are typically a lot more non-hit songs than there are hit songs.
Therefore, if the training and test songs would be totally randomly picked the resulting dataset would probably contain more non-hits than hits.
As one measure of the performance is accuracy and in this scenario a model is trained which will give a prediction of non-hit every time regardless of the input, would this in the sense of accuracy be a good model. Even though it is a total garbage.
To ensure that this does not happen the sample taken from the original data must represent equally non-hit and hit songs by amount.
To sample the data like this, another theoretical issue must be raised. As there are more non-hit songs than hit songs, how will the decision be made which non-hits songs to include and which not?
This kind of sampling process is called undersampling (Undersampling ref) and the key with the song data used in this work is to do the undersampling based on the dates.

2.7 Training & test split

As (Aurélien Géron 2017, p. 29) mentions a common practice is to split the data into training and testing sets and a common strategy is to use 80% as training and 20 % as test data.
The intuition for the validation and test set is to mimic data that the model has not “seen” before and measure the model’s performance on unseen data during training.

2.8 Over- & underfitting

As rounds of searching the right parameters for the model runs on, it could seem that the model is getting better the whole time.
However, usually at some point the model’s performance start to plateau and drop when trained too long. This phenomenon is explained in (Aurélien Géron 2017, p. 28) to be due to optimizing the model too much or overfitting the model to the data. Some models have hyperparameters which are trying to regularize the model to not overfit the data or slow the process at least down.
An opposite name is used when the model is not complex enough to have a good performance on the test or real-world data. Main methods to avoid underfitting is to use a more complex model or add the complexity.

2.8.1 Validation set & cross-validation

Cross-validation is a resampling method for machine learning evaluation and performance testing. It is recommended to be used when searching for hyperparameters. There are multiple implementations of the cross-validation. A version of it is k-fold validation.
In k-fold the dataset is split into a k number of test and training sets and the model is fitted with every set. The final accuracy is the average of accuracies among the iterations.

2.9 Models & algorithms

The models used in this work were the ones that can be seen in most of the papers on the same problem domain. (REFS!) Therefore, mostly the same reasoning applies here for this specific set of models. They are not on the same level of simplicity, and they do have differing methodologies to crack essentially the same problem.
Other contributing factor for choosing the models was previously recorded good performance on the papers of same problem domain.

2.9.1 Logistic regression

A logistic regression model computes a weighted sum of the input features (plus a bias term), but instead of linear regression, which would use the result as output, a logistic function is applied to the sum and that is used as output. When the classes cannot be linearly separable, logistic regression will fail. (Aurélien Géron 2017, p. 137-138; Ankur. A. Patel, 2019, 412).

2.9.2 Support Vector Machines (SVM)

A very powerful and capable machine learning model capable of performing multiple different machine learning tasks. The basic idea of SVMs in a binary classification set up is to find decision boundary, a plane which separates the two classes. There are different kind of ways to find a decision boundary and what works depends really on if the data is separatable in the way used. For example, linearly separable classes means that you can fit a straight line between the two classes in 2-dimensional space. (Aurélien Géron 2017, p. 147-167; Ankur. A. Patel, 2019, 484-500).

2.9.3 Neural Networks (NN)

A simple neural network architecture is a perceptron. A perceptron calculates a weighted sum of its inputs, applies a step function and outputs results. If a perceptron would have only one output and the step function used would be a sigmoid function, this would make the perceptron essentially the same as a logistic regression model. The difference is that the output layer can me scaled to be more than one and these layers can be stacked, making a multi-layer perceptron. A multi-layer perceptron model uses an activation function instead of step function to take advantage of backpropagation algorithms it is usually trained on. (Aurélien Géron 2017, p. 261-267; Ankur. A. Patel, 2019, 500).

2.9.4 Random Forest

A random forest model belongs in a group of ensemble methods. An ensemble method contains multiple predictors and takes an advantage of them all to make the final prediction. A group of decision tree classifiers trained with a random subset of the original data makes a random forest model. A decision tree predictor itself can be though as a logical set of binary questions on features in the data. Tree based models' biggest weakness is the tendency to overfit the data, but tree-based ensemble methods like random forest helps to keep the model more generalizable. (Aurélien Géron 2017, p. 183, 190, Ankur. A. Patel, 2019, 452-484).

3 Empirical part

Code & notes for the implementation: https://github.com/doslindos/billboard-ml-notes/blob/master/src/thesis_results.ipynb

3.1 The data collection

Steps defined here are found in the code source material from section “Collecting the data”.

3.1.1 Collect song information with Kaggle billboard data

The original source for the data is songs in the Kaggle billboard dataset (Kaggle, Billboard dataset, 2022). To retrieve the data, Kaggle python library is used, and the implementation can be found in **data/query/billboard.py**, function **downloadBillboardData**.
A billboard description is defined in **data/types/billboard.py** as **BillboardSong**.
The target data for machine learning algorithms is the audio features data from Spotify. To query it, the Spotify API needs the **Spotify song id** for the song that is queried.
The unique **Spotify song id** is not in **BillboardSong** information from Kaggle, which means that the queried hit songs can not be directly referenced via the API.
A matching process must be implemented. For every song, its name and artist name is used as a query string, when fetching data from Spotify search API.
The Spotify search API is returning songs that are close match, but there is a possibility that the song does not exist in the Spotify collection or is not available in the geographical location of the account.
To handle these cases, the closest returned result is compared with the query string.
If the query result artist name and song name is close enough to the artist’s name and song name in billboard list song that was used to make the search query, the song will be added to the final dataset.
Spotify’s collection contains a lot of “Instrumental”, “Karaoke”, etc. versions of popular songs. To keep the song data original, a song that has one of these words are discarded from the final dataset.
Fuzzywuzzy was used to for the matching process.
Spotipy was used to make the actual queries into the API and the implementation can be found in **data/query/spotify_api.py** function **getSpotifyDataFromBillboardSongsV2**.
A song queried from search api is defined in **data/types/spotify.py** as **SpotifySongInfo**.

3.1.2 Collect audio features with song data

SpotifySongInfo do contain the **Spotify song id** for the audio features API. In **data/query/spotify_api.py** function **getSpotifyAudioFeaturesV2** holds code to make the queries.
The audio features API can take total of 50 songs in one query, so first step is to batch the songs in sets of 50. The documentation informs that the order of songs should match the order of return values, but the resulting features are checked to hold the same **Spotify song id** than the **SpotifySongInfo**. The **SpotifySongInfo** and **SpotifyFeatures** are stored in a dictionary defined as **SpotifySongData**.
The collected features are defined as **SpotifyFeatures** and **SpotifySongData** in **data/types/spotify.py**.
The function returns a dictionary where the key is **Spotify song id** and value is a **SpotifySongData** object.

3.1.3 Collect non-hit songs

At this point all hit songs are collected with audio features. Collecting the non-hit songs is straightforward process. First every hit song is iterated, all tracks from the same album are queried from Spotify album API with **Spotify album id** and a random sample of 5 songs are selected from that album as non-hit songs in the final dataset. Every sampled song is checked not to be the original hit song that was made to query the album API.
After collecting **SpotifySongInfo** of the returned and sampled tracks, the audio features are collected the same way as defined in section above.
Code to this step is found in **data/query/spotify_api.py** function **getSongsWithAlbumsV2**.

3.2 Processing

The following sections define implementations of processing the data to be ready for consumption of the models. The notes can be found in source under “Song data preprocessing”.

3.2.1 Balancing & sampling the data

The total track amount in the final dataset is non-hit heavy with over three times more instances.

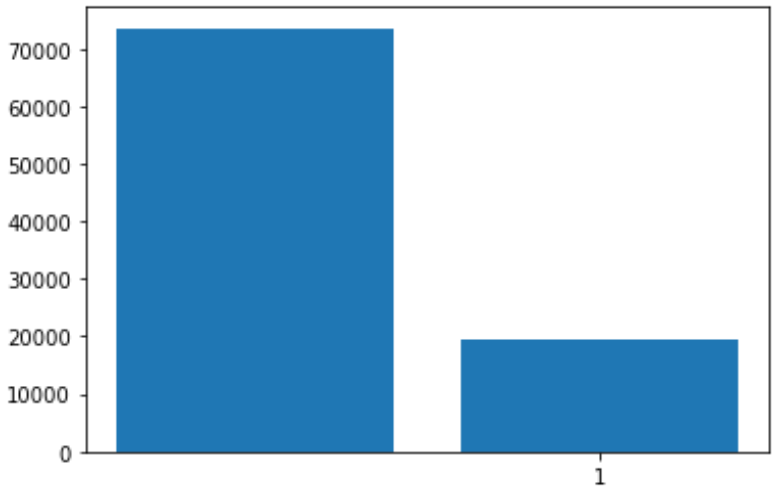

Figure 1: Total amount of non-hit (left) & hit songs (right) in source data used in this report

As you can see from the figure 1, the number of total non-hits is 73 562 and the number of hit songs is 19 462. The balancing of the data as described in theoretical part must consider two domains, the label and release year of the song. This is done by taking all hit songs by year and randomly sampling the non-hit song dataset by the same year, taking out the same amount as there are hit songs for that year. This should result a label- and release year-wise balanced dataset.

It is implemented in this work by looping over every unique release year of the hit songs that is between 1965 and 2021, then taking all hit and non-hit songs from the datasets of the specified year and sampling the resulting subsets. In year 1965 was smallest number of unique hit songs during the year (See figure 2).

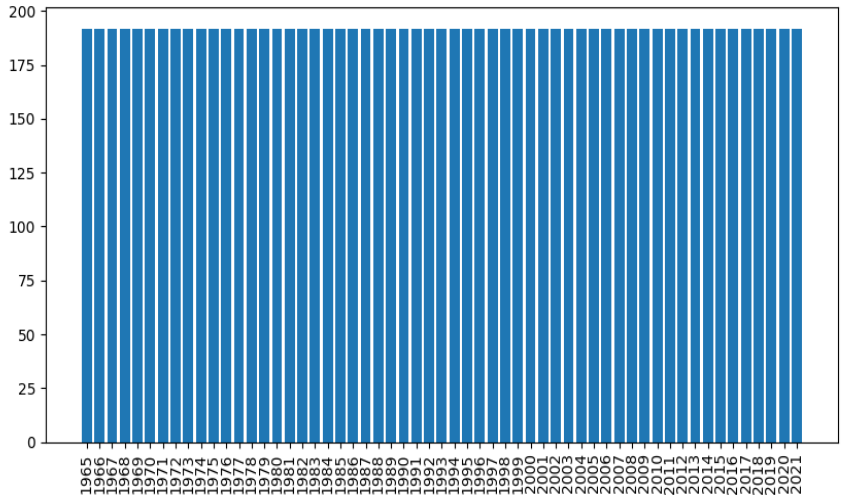

Figure 2 (By year)

This works as a sample size as all other years have at least that amount of unique hit songs released within. This balancing method gives the wanted results in label and time domain.
The implementation can be found in **data/process/balance.py** function **sampleByYears**.
The resulting balanced dataset can be seen in figure 3.

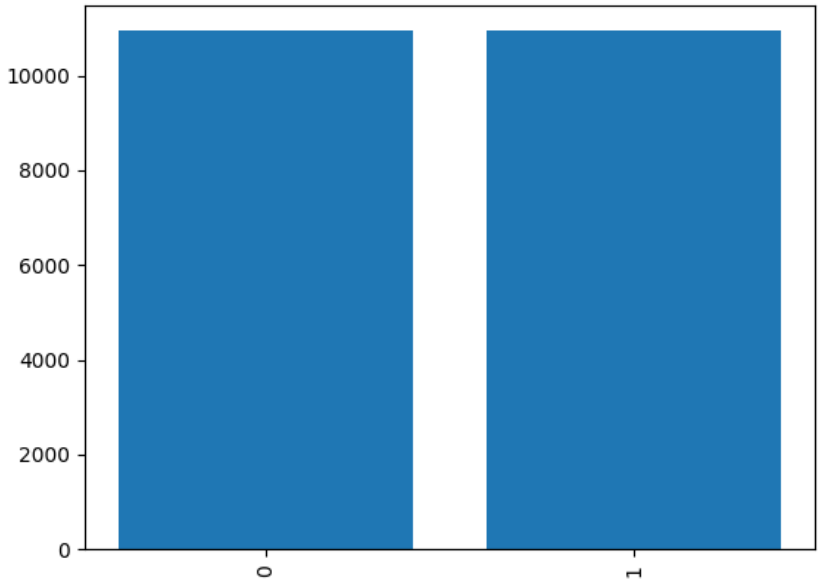

Figure 3 (Balanced dataset)

3.2.2 Transforming the data

In source material “Apply transformations” describe the implemented feature transformations mentioned in the theoretical part.
Firstly, the balanced datasets are concatenated as one (As in figure 4).

timeSignature	durationMS	key	mode	acousticness	danceability	energy	instrumentalness	liveness	loudness	speechiness	valence	tempo	year	label
4	197800	1	1	0.278	0.657	0.715	0.000146	0.0838	-6.965	0.0273	0.487	99.991	21	1
4	278467	0	1	0.127	0.611	0.479	0.000274	0.0777	-16.170	0.0364	0.696	124.369	21	1
1	193455	1	1	0.415	0.424	0.511	0.000000	0.7050	-9.609	0.4730	0.705	77.057	21	1
3	155627	10	1	0.662	0.505	0.299	0.000000	0.2450	-11.465	0.0276	0.722	81.338	21	1
4	190680	8	1	0.723	0.532	0.584	0.000000	0.1010	-5.254	0.0248	0.367	88.003	21	1

Figure 4 (Features)

In figure 4 is the features before any transformations are applied.

A scikit-learn librarys ColumnTransformer is used to specify which columns in the data-frame will be fitted in which transformer. Scikit-learn MinMax scaler & OneHotEncoder are used to implement the actual transformation part.

3.3 Models & hyperparameters

“Modelling” part describes the methods to initialization and execution of model training. In the initialization part, the cross-validation grouping helper function GroupKFold from scikit-learn is initialized. It is used to define the number of splits for cross-validation.
The data is also transformed.
For every model used, hyperparameters are searched with scikit-learn GridSearch. The range of different sets of values are given as a dictionary called *params*.
Finally, the accuracy of the best hyperparameter combination per model is printed out.
As you can see from table x, the best hyperparameters for Logistic Regression are x, for Support Vector Machine are x, Multi-Layer Perceptron are x and finally Random Forest are x. Full listing in table 1.

Hyperparameters			
Logistic Re-gresssion	Support Vector Machine	Neural Network	Random Forest
C: 0.1	C: 10	Activation: ’relu’	N_estimators: 800
L1_ration: 0.4	Class_weight: ’balanced’	Alpha: 0.0001	Max_features: ’auto’
Max_iter: 1000	Gamma: 0.1	Early_stopping: True	
Multi_class: ’ovr’	Kernel: linear	Hidden_layer_sizes: (20,)	
Penalty: ’elasticsearch’	Max_iter: -1	Max_iter: 1000	
Solver: ’saga’		Solver: ’adam’	
Warm_start: True			

Table 2: Best found hyperparameter set by model

The hyperparameter search took different amounts of time for different models. The times are specified in table 2. As you can see the Multi-Layer Perceptron and Random Forest took significantly longer time to search than Logistic Regression and Support Vector Machine.

Search time			
Logistic Regresssion	Support Vector Machine	Neural Network	Random Forest
~3	~5	~30	~20

Table 3: Hyperparameter search time in minutes

4 Results & discussion

Accuracies			
Logistic Re-gresssion	Support Vector Machine	Neural Network	Random Forest
0.55	0.55	0.54	0.56

Table 4: Overall model accuracies on the test set

As the table 4 shows, the model results were far from the impressive numbers recorder in some previous studies. To predict hit and non-hit songs for Billboard list featured artists seem to need more complex input features than just the audio features used in this work. Extensive analysis on the correlations between different features in this dataset with some proper feature engineering could yield in more flamboyant results.

Time used for hyperparameter search left hopes for improvement. It could be easily bettered by adding more parameters in the GridSearch param section and running the search again.

New models could also be introduced, although the ones that were chosen, had already proven track record for the same problem.

For the data gathering and processing part, an even more extensive methodology to group the songs, by balancing the songs also by genre or specific feature could produce more interesting results. Also, during the work an idea rose of splitting the dataset in sets by artist, but the implementation was shown to be to complex to try out.

In a more positive note of the yielded results by this work, a methodology of gathering Spotify audio features were introduced and made publicly available for anyone wanting to try out something similar with good amount of documentation.

The work is also modularized so that the source data of billboard songs are replaceable by basically any list of songs with a song and an artist name.

The public availability means that anyone with Kaggle and Spotify credentials can try out to get similar results or extend the work done here, for example with the suggested improvements.

The gathering methods did answer the purpose of this study, which was to understand what is available and to experiment with the audio features to find out interesting use-cases. During the implementation part of this work the availability and new potential usages beyond this experiment were reported to the assigner.

Sources

ISMIR 2021, *The International Society for Music Information Retrieval,* https://www.ismir.net/

MIREX 2021*, The Music Information Retrieval Evaluation eXchange,* https://music-ir.org/mirex/wiki/MIREX_HOME

David Moffat, David Ronan, Joshua D. Reiss 2015, *AN EVALUATION OF AUDIO FEATURE EXTRACTION TOOLBOXES,* *https://www.ntnu.edu/documents/1001201110/1266017954/DAFx-15_submission_43_v2.pdf*

The Echo Nest blog 2021*, https://blog.echonest.com/*

Darrell Etherington, Techcrunch 2014, *Spotify Acquires The Echo Nest, Gaining Control Of The Music DNA Company That Powers Its Rivals*, https://techcrunch.com/2014/03/06/spotify-acquires-the-echo-nest/
Read: 15.12.2021

Spotify API documentation*,*
Spotify search item, https://developer.spotify.com/console/get-search-item/
Spotify track audio features, https://developer.spotify.com/console/get-audio-features-track/

Spotipy, 2021, https://spotipy.readthedocs.io/en/2.19.0/

François Pachet & Pierre Roy 2008, *Hit song science is not yet a science,* https://ismir2008.ismir.net/papers/ISMIR2008_133.pdf

Matthias Mauch, Robert M. MacCallum, Mark Levy, Armand M. Leroi 2015, The Evolution of Popular Music: USA 1960-2010, https://arxiv.org/abs/1502.05417

Dorien Herremans, David Martens & Kenneth Sörensen 2014*, Dance Hit song prediction,* https://dorienherremans.com/sites/default/files/wp_hit.pdf

Kai Middlebrook & Kian Sheik 2019*, SONG HIT PREDICTION: PREDICTING BILL-BOARD HITS USING SPOTIFY DATA*, https://arxiv.org/pdf/1908.08609.pdf

Aurélien Géron 2017*,* Hands-On Machine Learning with Scikit-Learn, Keras, and Tensor-Flow: Concepts, Tools, and Techniques to Build Intelligent Systems, 1st edition, Kindle eBook

Ankur. A. Patel, 2019, Hands-On Unsupervised Learning using Python, ISBN 978-1-492-03564-0, Kindle eBook

Wikipedia 2021,
List of record charts, https://en.wikipedia.org/wiki/List_of_record_charts,
Digital audio, https://en.wikipedia.org/wiki/Digital_audio

Billboard 2018, *Billboard Finalizes Changes to How Streams Are Weighted for Billboard Hot 100 & Billboard 200,* https://www.billboard.com/pro/billboard-changes-streaming-weighting-hot-100-billboard-200/

Meghan Neal, Vice 2015, *A Machine Successfully Predicted the Hit Dance Songs of 2015* , https://www.vice.com/en/article/bmvxvm/a-machine-successfully-predicted-the-hit-dance-songs-of-2015

Adewale Adeagbo, 2020, *Predicting Afrobeats Hit Songs Using Spotify Data*, https://www.semanticscholar.org/paper/Predicting-Afrobeats-Hit-Songs-Using-Spotify-Data-Adeagbo/25484dca8f3a4fb1f65ae710f95dc895e7d18a5d

E. Georgieva, Marcella Suta & N. Burton, 2018, *HITPREDICT : PREDICTING HIT SONGS USING SPOTIFY DATA STANFORD COMPUTER SCIENCE 229 : MACHINE LEARNING*, https://www.semanticscholar.org/paper/HITPREDICT-%3A-PREDICTING-HIT-SONGS-USING-SPOTIFY-229-Georgieva-Suta/5f40e603ac969eb476f1af21b1efdd70153c24c0?p2df

Minna Reiman & Philippa Örnell, 2018 *Predicting Hit Songs with Machine Learning*, https://kth.diva-portal.org/smash/get/diva2:1214146/FULLTEXT01.pdf

Dorien Herremans & Tom Bergmans 2017, *Hit song prediction based on early adopter data and audio features*, https://dorienherremans.com/sites/default/files/paper_preprint_hit.pdf

Scikit-learn API 2021,
Documentation: https://scikit-learn.org/stable/modules/classes.html
Minmax scaler: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
OneHotEncoder: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html
Dataset splitter: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
Logistic Regression model: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Kaggle dataset, 2021, https://www.kaggle.com/dhruvildave/billboard-the-hot-100-songs