
**FOM Hochschule für Oekonomie & Management**

Hochschulzentrum Düsseldorf

**Scientific Paper**

part-time degree program

5th Semester

in the study course "Wirtschaftsinformatik"

as part of the course

Big Data & Data Science

on the subject

# Predicting Music Genres based on Spotify Song Data using a Gradient Boosting Algorithm

by

Thomas Keiser

Martin Krüger

Jesper Wesemann

Luis Pflamminger

| | |
|---|---|
| Advisor: | Prof. Dr. Adem Alparslan |
| Matriculation Number: | 123456 (Krüger), 123456 (Keiser),123456 (Wesemann), 123456 (Pflamminger) |
| Submission: | January 31st, 2022 |

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **API** | Application Programming Interface |
| **REST** | Representational State Transfer |

# List of Symbols

# 1 Einleitung

Dies soll eine LaTeX-Vorlage für den persönlichen Gebrauch werden. Sie hat weder einen Anspruch auf Richtigkeit, noch auf Vollständigkeit. Die Quellen liegen auf Github zur allgemeinen Verwendung. Verbesserungen sind jederzeit willkommen.

## 1.1 Problem Definition

## 1.2 Goal

## 1.3 Structure

# 2 Fundamentals

## 2.1 Basic Concepts of Big Data

Big Data is an umbrella term used to describe various technological but also organizational developments. Originally, Big Data refers to large sets of structured and unstructured data which must be stored and processed to gain business value. Today, Big Data is also often used as buzzword to outline various modern use cases that deal with large amounts of data. Big Data is therefore often used in conjunction with other buzzwords like automatization, personalization or monitoring. This chapter presents the foundation of Big Data in its technical implementation and combines the topics with busines cases.

### 2.1.1 Relevance of Data

Data in combination with Business Intelligence become increasingly important over the past decades and is closely associated with the advances of the internet itself. Looking back, Business Intelligence can be divided into three sub-categories, which follow another linearly. The first phase is centered around getting critical insights into operations from structured data gathered while running the business and interacting with customers. Examples would be transactions and sales. The second phase focuses increasingly on data mining and gathering customer-specific data. These insights can be used to identify customer needs, opinions and interests. The third phase, often referred as Big Data, enhances the focus set in phase two by more features and much deeper analysis possibilities. It allows to gain critical information such as location,person, context often through mobile and sensor-based context.

In conclusion, organizations require Business Intelligence as it allows them to gain crucial insights which is needed to run the business and achieve an advantage over the competition. It is important to minimize the uncertainty of decisions and maximize the knowledge about the opportunity costs and derive their intended impacts. It is clearly noticeable that the insights and analysis possibilities become progressively deeper and much more detailed. Along this trend the amount of data required becomes larger and larger with increasingly complex data structures. Size, complexity of data and deep analysis form the foundation of Big Data and can be found again in the 5V matrix of Big Data.

### 2.1.2 The 5V Matrix for Big Data

When describing Data, a reference is often made to the five Vs, which highlight its main characteristics. The previous aspects of Big Data can again be recognized in averted form.

**Volume:** The size of the datasets is in the range of tera- and zettabyte . This massive volume is not a challenge for storing but also extracting relevant information from the mass of data.

### 2.1.3 Reinforcement Learning

### 2.1.4 Machine Learning Algorithms

## 2.2 Gradient Boosting

## 2.3 Cross Industry Standard Process for Data Mining

## 2.4 Application Programming Interfaces

This section gives an overview over the basic concepts[1, S.1] and technologies behind Application Programming Interfaces (APIs).

### 2.4.1 Purpose and Usage

An APIs is an interface between two pieces of software.[1, S.1] These might run on the same machine and communicate locally, in the case of a desktop application for example, or on seperate machines that are connected via some network, e.g. in a client/server application.

APIs provide a readily implemented solution to a problem in programming and can be reused , how the API can be used to solve it. APIs such a problem might be finding some value in an array, fetching a file from a hard drive, or getting the latest weather data from a weather service. **@expl@@@filehook@file@pop@assign@@nnnn** sections/02_Fundamentals05_api.texsections/02_Fundamentals05_api.tex

## 2.5 Basic Concepts of Music Theory

# 3 Implementation

## 3.1 Data Collection

In this section the approach and inplementation of data collection for this project is examined.

### 3.1.1 Requirements for the dataset

Basic requirements the dataset should fulfill are

- **Includes Spotify song features**

  Spotify provides a set of song features that were generated using their own models. The dataset should inlude these features, as they are needed to train the model

- **Includes genre as label**

  The dataset needs to include the genre of the track to use as a label for the classifier

- **Has sufficient sample size per genre**

  In order to train the model well, a sufficient sample size is needed per genre. It was not known before collecting the data, how many samples were enough.

- **Song and Artist name**

  The best way to filter out duplicates is to use the song and artist names. Spotify does provide a track id for each song, however, if a song is released twice (e.g. as a single and later in an album), these track ids will differ which will lead to a duplicate entry.

Additional fields are not going to be used in this analysis, but might still be collected in order to publish the dataset and enable others to use it for different applications.

### 3.1.2 Existing Datasets

As this paper examines creating a model specifically on Spotify Song Data, a search on the internet was conducted first, to find a potential pre-made dataset, pulled from the Spotify API, which could be used. Kaggle [1] lists an extensive catalogue of community provided datasets, so the main sources of this search were Kaggle and Google search for the term "Spotify Song Data". Kaggle lists a couple of datasets that could be applicable to the research question in this paper. Some examples of datasets listed are given and explained, why they could not be used for this project.

"Spotify music analysis" by user Aeryan [2] is a dataset of 2017 rows, which includes musical features like acousticness and tempo, the song title and artist, but lacks a genre field. Because of the small sample size and the missing genre field, this dataset could not be used.

There are multiple datasets which include songs that were featured in Spotify's "Top 50" Playlists, charts, or year in review, recorded at a single point in time or historically. [3] [4] These could not be used, as the sample size is again too small and the focus is specifically on the most popular tracks and not a wide variety of music in a genre.

"Dataset of songs in Spotify" [5] is the most promising dataset examined, as it has a big sample size and includes genre data. However, the methodology of how the data was collected is not included and there could be multiple ways of how genre data for a given song is collected, as is explained later. Also the genres are limited to very specific directions of Electronic Dance Music and Hip-Hop.

As no optimal dataset for this research paper could be found using our search criteria, a dataset was specifically created for this paper using the Spotify Web API.

### 3.1.3 Ressources and Approach

Spotify provides extensive documentation for developers on their developer website [6]. This includes development and design guidelines for teams, that want to integrate Spotify's service into their own apps, documentation on IOS and Android development a community forum, a developer dashboard and the Web API documentation, which is the main ressource for data collection from Spotify.

[1] Kaggle Website: https://www.kaggle.com/
[2] https://www.kaggle.com/aeryan/spotify-music-analysis
[3] https://www.kaggle.com/nadintamer/top-spotify-tracks-of-2018
[4] https://www.kaggle.com/leonardopena/top50spotify2019
[5] https://www.kaggle.com/mrmorj/dataset-of-songs-in-spotify
[6] https://developer.spotify.com/

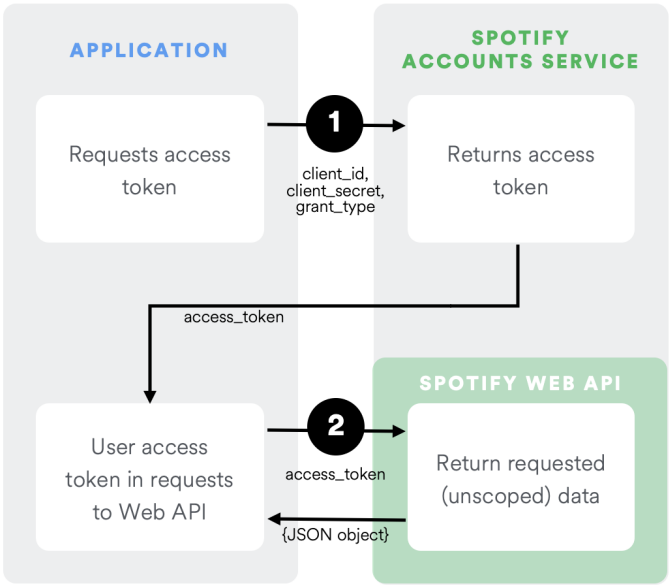
The API is based on the Representational State Transfer (REST) architecture. The different endpoints return JSON metadata directly from the Spotify Data Catalogue [2]. There are also features to query for user related data using an authorization flow with the users Spotify account, but this is not relevant in this context. [2] Requests to the API are made via HTTPS GET or POST methods. The API can be used by anyone, but authorization via the OAuth protocol is required to access data from the API. To explore the API and find endpoints to use, Spotify provides a developer console, which can be used to send requests and see what kind of responses come back. This is not suitable for saving the data or making multiple requests programmatically, but is helpful for API exploration. As there is not one single endpoint that delivers all required fields, multiple queries that build on top of each other have to be made.

The approach began with using the API reference to get an overview over the endpoints and their responses. The specific endpoints that might return interesting data were queried using the Spotify Web Console, to see a response with live data and which exact fields are returned. Beyond the Web Console, the tool "Postman" was used to explore the API. It is a platform that can be used to make HTTP requests to an API and store these requests in a collaborative environment. [3] It supports the required authorization workflows and enabled the research team to explore endpoints together, easily make API calls without having to authenticate by hand, and save the endpoints and required input parameters in a shared workspace. Once exploration was complete, the complete data collection was implemented in Python 3, mainly using the libraries http, json and requests. The final result was saved as a CSV file to be used for data exploration and further processing.

### 3.1.4 Authorization

Using the Spotify documentation for authorization workflows [4], authorization was first tested using Postman and then implemented in Python. The workflow is explained using the Python code. Using a Spotify account to log in, the developer dashboard can be accessed. Here an application was registered with Spotify for the research project. Spotify tracks API usage per application and can recognize if the API is being abused or too many requests are sent, which will result in rate limitation or blocking from the API. On the API Dashboard, a "Client ID" and "Client Secret" can be retrieved. These credentials are used to start the authorization flow, as described by Spotify in Figure 1.

**Figure 1: Spotify Authorization Flow**




Source: [4]

Step one is a post request to the Spotify Account Service with the client id and client secret from the application dashboard, which returns an access token, that is valid for one hour. This token can be used to access any endpoint of the actual API that does not require user specific data. When the token expires, a new one has to be requested before querying the API again. Figure **??** shows the full request and response to acquire the token.

**Figure 2: Access Token Request**

**POST** **https://accounts.spotify.com/api/token**

*request access token*

**Body** application/x-www-form-urlencoded

```
{
    "grant_type": "client_credentials",
    "client_id": client id from application dashboard,
    "client_secret": client secret from dashboard
}
```

**Response** application/json

**200** ok


```
{
    "access_token": "BQDgQCSx-tIMDo9LfVeZxm6YMl2p_WbEU3Q
        9ENsVl7e--6d_vockTsfzMVUhPWiHSSnFUuHvm_9POA1kYEw"
        ,
    "token_type": "Bearer",
    "expires_in": 3600
}
```


In the Python implementation, the requests library is used to execute the request in figure **??** and store the access token in a variable. The dotenv library is used to read the client id and secret from a seperate .env file, rather than writing it into the code. This prevents these sensitive credentials from being committed into version control, which is hosted in a public GitHub repository and would therefore make the credentials public.

```
#Get environment variables from ".env" file and read credentials
load_dotenv('.env')
client_id = os.environ.get('CLIENT_ID')
client_secret = os.environ.get('CLIENT_SECRET')

# Authenticate and get an API Token from Spotify using a Client ID and secret
def getAuthTokenFromCredentials(id, secret):

    url = "https://accounts.spotify.com/api/token"

    payload = f'grant_type=client_credentials&client_id={id}&client_secret={secret
        }'
    headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    }

    response = requests.request("POST", url, headers=headers, data=payload)

    return response.json()["access_token"]

auth_token = getAuthTokenFromCredentials(client_id, client_secret)
```

### 3.1.5 Getting Features

In order to predict the genre of a track based on audio features, these features have to be requested for every track. Spotify provides an endpoint to get audio features for a single track or up to 50 tracks at a time. The latter is used in the Python implementation as it

reduces the number of requests to be made. The typical request/response pattern for the audio-request endpoint of a single track is shown in figure 3.

**Figure 3: Audio Feature Request**

**GET** **https://api.spotify.com/v1/audio-features/{id}**
*request audio features for id*

**Parameter**

id id of the song

**Response** application/json

**200** ok


```
{
    "danceability": 0.677,
    "energy": 0.638,
    "key": 8,
    "loudness": -8.631,
    "mode": 1,
    "speechiness": 0.333,
    "acousticness": 0.589,
    "instrumentalness": 0,
    "liveness": 0.193,
    "valence": 0.435,
    "tempo": 82.810,
    "type": "audio_features",
    "id": "2e3Ea0o24lReQFR4FA7yXH",
    "uri": "spotify:track:2e3Ea0o24lReQFR4FA7yXH",
    "track_href": "https://api.spotify.com/v1/tracks/2e3
       Ea0o24lReQFR4FA7yXH",
    "analysis_url": "https://api.spotify.com/v1/audio-
       analysis/2e3Ea0o24lReQFR4FA7yXH",
    "duration_ms": 211497,
    "time_signature": 4
}
```


With the exception of type, id, uri, track_href and analysis_url, all of the fields included in this response can be used as features in the dataset. However, this api call expects a track id, which we need to get using other api calls first. This could be a search endpoint, getting all tracks in a playlist, etc. Also, it does not give the track or artist names and doesn't include a genre.

### 3.1.6 Getting Track IDs and Labels

There is no simple endpoint that takes one or more track ids and returns a "genre" field in its response. The exploration of the API using the reference, web console and Postman only revealed two ways of getting the genre of a track.

The first way is using the artist of a track. Given a track id, the artists of the track and their corresponding ids can be requested by using the "/tracks/id" endpoint. Then, using the artist id, the genres that an artist is known for are returned, as can be seen in figure 4.

**Figure 4: Artist Request**

**GET** **https://api.spotify.com/v1/artists/{id}**
*request information about an artist by their id*

**Parameter**

id id of the artist

**Response** application/json

**200** ok

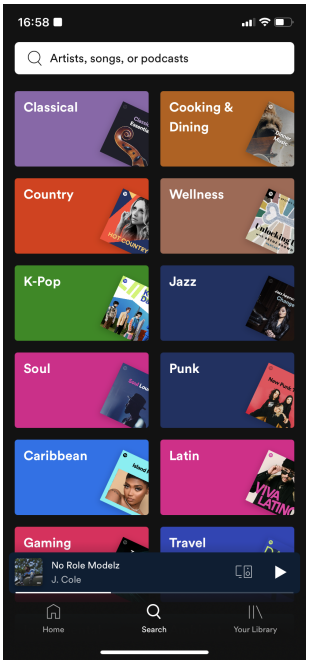

```
{
    "external_urls": {
        "spotify": "https://open.spotify.com/artist/6l3
           HvQ5sa6mXTsMTB19rO5"
    },
    "followers": {
        "href": null,
        "total": 15554811
    },
    "genres": [
        "conscious hip hop",
        "hip hop",
        "north carolina hip hop",
        "rap"
    ],
    "href": "https://api.spotify.com/v1/artists/6l3HvQ5
       sa6mXTsMTB19rO5",
    "id": "6l3HvQ5sa6mXTsMTB19rO5",
    "images": [
        ...
    ],
    "name": "J. Cole",
    "popularity": 89,
    "type": "artist",
    "uri": "spotify:artist:6l3HvQ5sa6mXTsMTB19rO5"
    }
```


This examplary API response shows a problem with this approach. One artist can be sorted into multiple genres. A given track might be associated with either of the artists genres, but the data does not show, which one exactly. Additionally a track might have multiple artists which further complicates this. Given these circumstances, this approach is problematic.

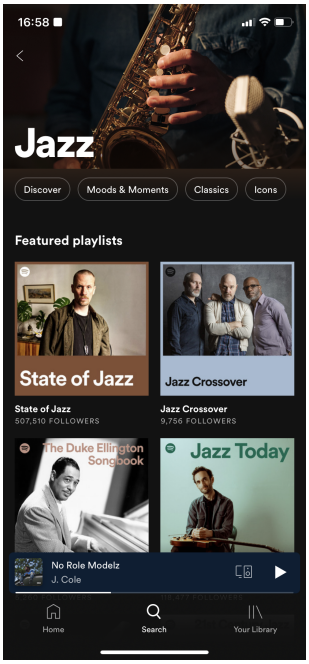
The second way is Spotifys "categories" feature. The apps search tab provides a number of categories that a user can browse through to find new music in their preferred genre or style. In figure 5a an overview over some of the categories that are available in the Spotify App is shown. There are categories of multiple types, e.g. specific activities, like Cooking or Gaming, or places, like "At Home" or "In the Car". But there are also categories for nearly all major genres. In the app screenshot there is for example Classical, Jazz or Soul. These categories can be used to get tracks that belong in each specific category. When a user taps on a one of the categories, playlists that contain tracks of the respective category are shown to the user, like shown in figure 5b. The API mirrors the apps behaviour and provides an endpoint to get a list of categories and their ids, one to get all playlists and playlist_ids in a category, and one to get all tracks and track_ids in a playlist. This chain of API calls is used to request every track in every playlist in a certain category.

**Figure 5: Categories and Playlists in Spotify App**

**(a)** Category Overview




**(b)** Playlists in category



### 3.2 Data Understanding

### 3.3 Data Preparation

### 3.4 Modeling

### 3.5 Evaluation

## 4 Fazit

# Appendix

## Appendix 1: Beispielanhang

Dieser Abschnitt dient nur dazu zu demonstrieren, wie ein Anhang aufgebaut seien kann.

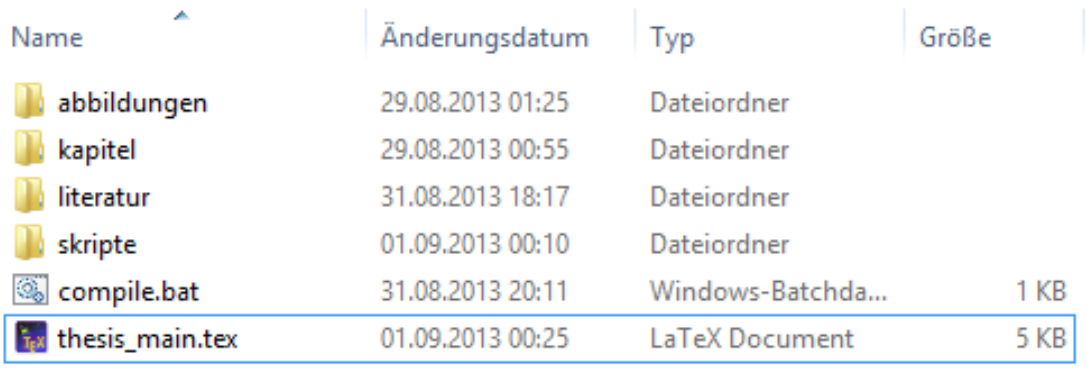
### Appendix 1.1: Weitere Gliederungsebene

Auch eine zweite Gliederungsebene ist möglich.

## Appendix 2: Bilder

Auch mit Bildern. Diese tauchen nicht im Abbildungsverzeichnis auf.

**Figure 6: Beispielbild**



# Bibliography

[1] M. Reddy, *API Design for C++*. Elsevier Science, 2011, ISBN: 9780123850041. [Online]. Available: https://books.google.de/books?id=IY29LyIT85wC.

## Internet sources

[2] ‘Spotify web api documentation,’ Spotify AB. (), [Online]. Available: https://developer.spotify.com/documentation/web-api/.

[3] ‘What is postman?’ Postman, Inc. (), [Online]. Available: https://www.postman.com/product/what-is-postman/ (visited on 2022-01-16).

[4] ‘Spotify authorization documentation,’ Spotify AB. (), [Online]. Available: https://developer.spotify.com/documentation/general/guides/authorization/.

## Declaration in lieu of oath

I hereby declare that I produced the submitted paper with no assistance from any other party and without the use of any unauthorized aids and, in particular, that I have marked as quotations all passages which are reproduced verbatim or near-verbatim from publications. Also, I declare that the submitted print version of this thesis is identical with its digital version. Further, I declare that this thesis has never been submitted before to any examination board in either its present form or in any other similar version. I herewith agree/disagree that this thesis may be published. I herewith consent that this thesis may be uploaded to the server of external contractors for the purpose of submitting it to the contractors' plagiarism detection systems. Uploading this thesis for the purpose of submitting it to plagiarism detection systems is not a form of publication.

Düsseldorf, 16.1.2022

(Location, Date)

(handwritten signature)