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 Decision Trees

Decision Trees are one of the most widely used supervised Machine Learning Algorithms either as standalone solutions or in combination with enhancement approaches like Boosting. They are furthermore very flexible in their construction and can be used for various Machine Learning Problems such as Classification and Regression.

Decision Trees “predict an unknown value of a target variable by learning decision rules from data features” to reconstruct the dependence between the features and the respective labels for each sample. To perform the Classification or Regression, Decision Trees rely on recursive splitting of the dataset into multiple subgroups. As the number of iterations increases, the subgroups become more and more homogeneous. The ideal result is that each subgroup is fully homogeneous and therefore only represent a single category (in case of Classification). However, this is often only a theoretical best condition, as multiple risks, such as overfitting, are associated with the increasing depth of Decision Trees.

Trees consist out of four main components. A Node is a discrete Decision Function that takes samples as its input and splits them based on features into subgroups. The aim of each split, as previously discussed, is to create a split that results in the overall most homogeneous distribution for all subgroups. Nodes can be subclassified into three kinds. The Top-Node, from which the classification starts, is called a Root Node. Nodes that are located at the very end of a Decision Tree are referred to as Leaves. Leaves do not split data any further and only mark the end of a Decision Tree. When reached, Leaves categorize or predict a final output value depending on the prediction task. Nodes in-between the Root Node and Leaves are called Internal Leaves. Like the Root, Internal

Leaves are responsible for the recursive splitting of the data. Branches connect Nodes with another. For classical Trees, information only flows from the top to the bottom of the Tree.

2.2.1 Decision Tree Algorithm

In practice, there exist various Algorithms for computing Decision Trees with the most common ones being: ID3, C4.5, C5.0 and CART. Each algorithm follows the same principle of regressively finding perfect splits to separate data but utilizes different methods to find the ideal splitting criteria, which strongly influences the structure of the Tree, its accuracy and performance. Additionally, each Algorithms has its benefits and constraints. Therefore, it is important to determine the best Algorithm before implementing the Decision Tree based on the prediction task and dataset. This project uses the Python library Sklearn to implement a Classification Tree. Sklearn is based on the CART Algorithm.

To better visualize the procedure of the Decision Tree Algorithm, a simplified dataset is used, on which the individual steps are explained. For this example, a Classification Problem is chosen. The dataset consists out of actual features and labels from the project implementation. The features are “acousticness” and “danceability”. Both features are numerical with a value range in-between 0 and 1. The Classification Problem is binary with "Hiphop" and "Jazz" representing the classes k for which the samples of the dataset are classified. Mathematically, the dataset is represented in the following form: Xi present the explanatory features while Yi represents the corresponding label for one data point of the input dataset N with a total number of i samples.

(TABLE WITH DATA)

(COORDINATE SYSTEM WITH DATA)

With the initialization of the dataset complete, the implementation of the Decision Tree Algorithm can begin. The Decision Tree Algorithm splits a Node represented by Q_m with N_m samples into multiple subgroups. The split can be binary, which means that Q_m is divided into two subgroups Q_m^{left} and Q_m^{right}, or multiway. While multiway splitting seems to be more advanced with greater prediction potential, in practice and for CART binary splitting is used almost exclusively. The split $\theta = (j, t_m)$ consists out of a feature j and a threshold t_m on which the division takes place. The Output $G(Q_m, \theta)$ is mathematically defined as the following:

$$G(Q_m, \theta) = \frac{N_m^{left}}{N_m} H(Q_m^{left}, \theta) + \frac{N_m^{right}}{N_m} H(Q_m^{right}, \theta)$$

The quality of the split is defined using an Impurity Function H. The Impurity Function has one Leaf Q_m^{left} or Q_m^{right} and the spliting criteria θ as its input. The best overall Gain $G(Q_m, \theta)$ is reached, if $G(Q_m, \theta)$is minimized (3).

$\theta^* = argmin_\theta G(Q_m, \theta)$

The most common Impurity Function H for Classification is Gini Index (4) and also used for CART Decision Trees. Gini index measures the probability that a sample does not belong to the category that represents the majority of the subgroup. If both categories of a subgroup are identical in size, the Gini Index reaches its maximum point at $0,5$ (5). The maximum of the Gini index means the worst possible data constellation for a subgroup. Gini index equal to 0 means on the other hand the best possible result with the subgroup being fully homogenous. p_i represents that probability that a sample belongs to the class j of k classes in total.

(4) GINI INDEX FORMULA

$Gini = 1 - \sum_{j=1}^{k} (p_j)^2$

(5) FORMULA AS A GRAPH

For the example the splits look like the following. The split of numerical features is more complicated as it is for categorical or binary features since can take it can take place at any value within the value range. Therefore, each value of the sample could be a possible threshold. To achieve the best overall Gini Index, it must be calculated for every possible threshold of every feature. The calculation below only shows the best possible splits for each feature according Gini Indexes. With G calculated for both features, the first split can be determined. When comparing both features it is visible that Danceability minimizes G more than Acousticness does and therefore is according to (3) determined as the overall best possible split for the Root Node.

Acousticness:

$Gini^{left} = 1 - ((\frac{2}{5})^2 + (\frac{3}{5})^2) = 0,48$

$Gini^{right} = 1 - ((\frac{3}{3})^2 + (\frac{0}{3})^2) = 0$

$G(Q_m, \theta) = \frac{5}{8} * 0,48 + \frac{3}{8} * 0 = 0,30$

Danceability:

$Gini^{left} = 1 - ((\frac{4}{4})^2 + (\frac{0}{4})^2) = 0$

$Gini^{right} = 1 - ((\frac{1}{4})^2 + (\frac{3}{4})^2) = 0,36$

$G(Q_m, \theta) = \frac{4}{8} * 0 + \frac{4}{8} * 0,36 = 0,18$

The splitting process is repeated for each subgroup until a stop-criteria is met. The natural stop criterion is a completely homogeneous dataset for a node and can also be found in the example for Leaf X. Such Internal Nodes automatically become Leaves and represent the category of samples. Other stop criteria can be predefined depending on the Algorithm used for implementation. The most relevant criterion is the definition of a maximum depth of the Decision Tree. Other hyperparameters are discussed in the implementation chapter.

Tree optimization plays a very relevant role because Trees often overclassify training data without countermeasures. Although the training data is well classified, overfitted Decision Trees often produce very poor results for test data. Too accurate classification of training data can negatively affect Decision Trees, as they are less able to generalize the learned knowledge. Pruning is a technique used to overcome overfitting problems by reducing the size of Decision Trees. Sections that provide little to no classification benefit are removed or not constructed during the recursive splitting process. In essence, worse results for training data are traded for better results for unknown data.

The experimental dataset is not large enough to take appropriate countermeasures. The complete Classification Tree is shown in figure X. For the second iteration, only the left subgroup was further split. The result are two fully homogeneous sub-subgroups created from the data Nm of the left subgroup. Because only two features were used, each split can be visualized using a coordinate system (figure X2). the colored areas present the respective splits.

2.2.2 Evaluation of Decision Trees

In conclusion, Decision Trees can be assessed as follows. Starting with the advantages, the main benefit is the overall simplicity of Decision Trees, both from a technical and business point of view. For researchers and developers, Trees are easy to construct, require little to no data preparation, are almost univerally applicable with a possibility of validation. However, the simplicity for business should not be underestimated either. When comparing Machine Learning Algorithms, the main comparison is often the accuracy of a model. The areas in which decision trees stand out include visualization and comprehensibility. The Decision Tree Algorithm is a white box model that allows complete transparency and explainability.

The disadvantages of Decision Trees are again closely related to its simplicity. Overfitting and the relative instability of Decision Trees are the main drawbacks and result in good momorization but a comparatively weak generalization ability.

2.3 Gradient Boosting

The Gradient Boosting Algorithm is derived from Gradient Boosting Machines, which are a family of powerful Machine Learning Algorithms with a certain procedure pattern for the creation of models. In general, GBMs are very flexible in their characteristics with the possibility of utilizing multiple different Machine Learning Algorithms as their foundation.

Boosting differs from classical approaches as it does not consist out of a single predictive model but an ensemble approach. Ensemble Algorithms contain multiple Weak Learners that form a committee to create a strong prediction. Weak Learners are often very simple forms of traditional Algorithms, like Decision Trees, and must just be able to predict parts of the dataset correctly. Only the combination of many Weak Learners allows the model to perform overall accurate predictions. The most common form of Ensemble Algorithms are Bagging Algorithms with Random Forests as an example. Bagging, in essence, is the combination of multiple unique models. The prediction is formed by aggregating the outputs from all models into a single representative value. Typically, all models are derived from a single Algorithm, like Decision Trees for Random Forests, but technically there is no limitation to aggregate outputs from different Algorithms. This is also the case for Boosting.

Boosting, on the other hand, follows a different principle and does not rely on independent models with an aggregation function. Boosting fits new models sequentially and can thereby use earlier acquired knowledge for further iterations. This allows GBMs to train specific areas of the dataset where it has previously performed poorly.

2.3.1 Gradient Boosting Algorithm

(1) GRADIENT BOOSTING ALGORITHM

The generic gradient boosting algorithm is shown in Figure X. It follows a sequence of three distinct steps, with step two being performed for each iteration. At the beginning, an additional initiation of the dataset and a Loss Function is necessary. The mathematical representation of the dataset is like the one used for Decision Tees. A summary: x_i present the explanatory features while y_i represents the corresponding label for one data point of the input dataset N with a total number of n samples.

The mathematical goal of the Algorithm is to reconstruct the unknown functional dependance f between x_i and y_i with an estimate $F(x)$ for every data point, such that the specific Loss Function is minimized (1).

(1) GRADIENT BOOSTING OPTIMIZATION

The Loss Function is an indicator for the quality of the model. A small Loss for a data point means that the prediction is close or identical to the observed Label and the model therefore categorizes the sample correctly whereas a high Loss implies that the model could not predict the sample well. Given a particular learning task and dataset, different Loss Functions must be considered as Loss Functions are only suitable for specific constellations. The most common Loss Function for Binary Classification is the so-called Bernoulli Loss (2). The Bernoulli Loss can be transformed into a log(odds)-prediction (3) as it is better suited for further calculations. Variations of (3) will be used in the following section to demonstrate the Gradient Boosting Procedure.

(2) BERNOULLI LOSS

(3) BERNOULLI LOG(ODDS)-PREDICTION

Additionally, a Machine Learning Algorithm must be defined as a Weak Learner. For GMB there are multiple Leaners to choose from, again the choice is mostly depending on the prediction task and available data. A classical approach is the use of Decision Trees, which was also chosen as the Weak Learner for this project. Decision Trees used for Gradient Boosting are always Regression Trees, regardless of whether they are used for Regression or Classification Problems. The optimization parameters are almost identical as for standalone Decision Trees, but the Trees often look very different because they are specifically created as Weak Learners. As a result, the Decision Trees often only consist out of very few levels with only 8-32 Leaves.

Algorithm Procedure

To showcase the Gradient Boosting Algorithm the same sample dataset is used as for Decision Trees. It again consists out of 8 samples with two features with two categories as target Labels.

DATASET

The first step 1) is to set an initial prediction for all samples of the dataset. The initial prediction is not unique for individual samples but a uniform value. The optimal initial prediction can be calculated using the following equation (3). For Fo(X), representing the initial prediction, a minimum of gamma is searched for. The right-hand side of the equation only consists out of a sum for each sample i (of the total dataset n) of the known Loss Function with the respective Label y1 and gamma as its input. To find the low point of the equation, the derivate of the Loss Function is required. The final calculation of the overall log(odds) that as song is classified as “hiphop” is the loge of the sum of songs of the category “hiphop” divided by the sum of the songs of the category “jazz” (4). The result

can be checked graphically as is equal to the X1 value of the low point of the log(odds)-prediction (5).

(4) F0(X) = log(L = 1/L = 0)

(5) COORDINATE SYSTEM

Finally, the log(odds)-prediction is converted back into a probability with the help of a Logistic Function (6). The result is that all songs have the probability of X to belong to the category “hiphop”.

(6) LOGISTIC FUNCTION

With the completion of step 1), one can start with step 2) of the algorithm. 2) is a sequential process of constructing the Regression Trees with a total of M iterations. The first iteration starts with m = 1.

In a) the Pseudo Residuals rim for each sample i of the dataset are created. The equation (7) for calculating the PR consists out of known fragments. For every sample i = 1, . . . , n a PR rim is calculated using the derivative of the log(odds)-prediction with the Label yi and the Prediction of the last iteration (f = fm – 1) as its input. Again, the equation can be transformed into a very simple equation (8). For each sample the PR can be calculated by only subtracting the previously calculated probability from the observed Label. Ideally, an additional column is created in which the Pseudo Residuals are temporarily stored (figure 9).

(7)

(8)

FIGURE PSEUDO RESIDUALS

b) constructs the Regression Trees put of the features of the samples with the corresponding PR as the label. For this example, only Tree Stumps are created to simplify the implementation. The Regression Tree for the first iteration is shown in Figure 10. After the completion of the Tree, Terminal Regions Rjm must be defined for every Leaf. J starts with 1 and is increased for every Leaf.

FIGURE REGRESSON TREE

Following the completion of the Regression Tree, Output Values gammaj,m are calculated by using the equation presented in (11). For each Leaf in the Tree gammaj,m is computed by finding gammaj,m that minimizes the equation (11). Like for the initialization step, the

derivative has to be created and must set equal 0. And again, after a complicated transformation, a very simple equation remains (12). The Output Value can be calculated using only the residuals and the most recent predicted probabilities for all samples in the Leaf.

(10)

(11)

Part d) marks the end of the first iteration and creates a new prediction F1(X) for each sample (13). The new prediction is based on the last prediction plus the Learning Rate V multiplied by the Output Value(s) for the sample of the last Regression Tree (14). Normally, there is only one Output Value for a sample which makes the summation sign obsolete. The Learning Rate is a hyperparameter for Gradient Boosting. For this example, a high V is used to better visualize the changes. In practice a V in the order of 0.1 is common.

(13)

(14)

Step 2 is repeated until M is reached. M marks the completion of the training of the Gradient Boosting Model. The FM(X) is the final prediction for every sample. For the final predictions thresholds are used to compute the category to which the samples belong. A typical threshold for Binary Classification is 0.5 as it splits FM(X) into two equal classes. This process also takes place for unknown samples.

Step 3) is the final step for Gradient Boosting and classifies unknown. An unknown sample gets initialized by Fo(X) and is sequentially routed through every model. For each iteration the Prediction is updated using the Output Value of the Regression Tree. Finally, the last FM(X) is used to classify the sample using the predefined threshold.

2.4 Cross Industry Standard Process for Data Mining

2.5 Application Programming Interfaces

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

2.5.1 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/0

2.6 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/

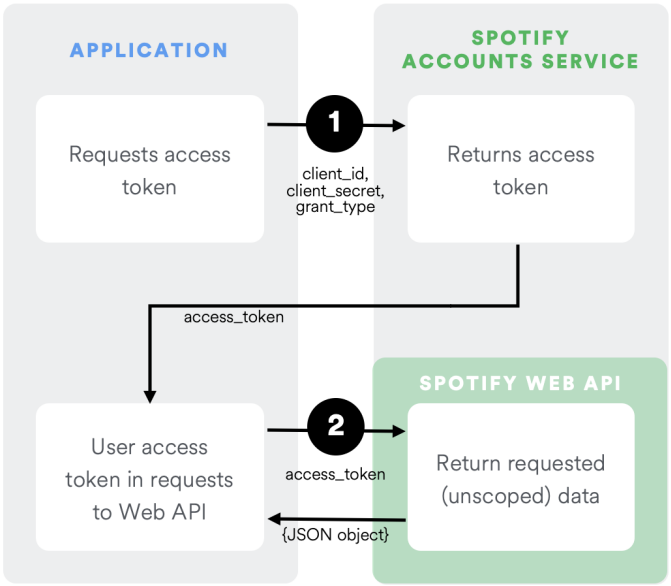
The API is based on the Representational State Transfer (REST) architecture. The different endpoints return JSON metadata directly from the Spotify Data Catalogue [2]. There are also features to query for user related data using an authorization flow with the users Spotify account, but this is not relevant 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

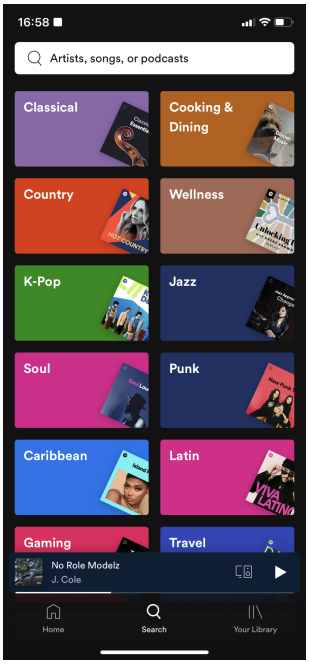

```
{
    "external_urls": {
        "spotify": "https://open.spotify.com/artist/6l3
           HvQ5sa6mXTsMTB19rO5"
    },
    "followers": {
        "href": null,
        "total": 15554811
    },
    "genres": [
        "conscious hip hop",
        "hip hop",
        "north carolina hip hop",
        "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.

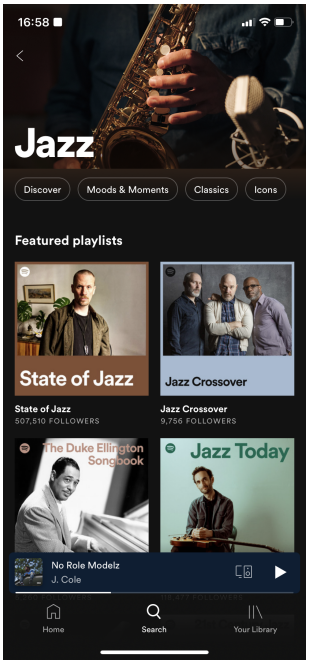
The second way is Spotifys "categories" feature. The apps search tab provides a number of categories that a user can browse through to find new music in their preferred genre or style. In figure 5a an overview over some of the categories that are available in the Spotify App is shown. There are categories 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.

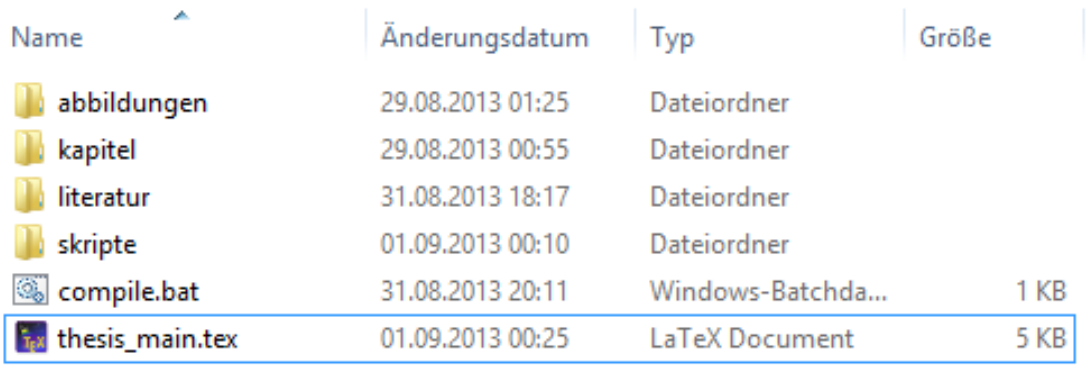
Appendix 1.1: Weitere Gliederungsebene

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=IY29LylT85wC.

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, 21.1.2022

(Location, Date)

(handwritten signature)