Spotify Million Playlist Challenge

Pablo Schneider - Malte Schwenker - Isa Kilic

02. März 2022

Inhaltsverzeichnis

1 Spotify Million Playlist Challenge

Die Spotify Million Playlist Challenge bezieht sich darauf, per datamining bzw. machine-learning Songs vorschlagen zu können, um Playlists zu vervollständigen. Hierfür gibts sowohl Algorithmische, als auch Logische Lösungswege.

1.1 Similarity Matrix

Track-Track similarity Matrix, in welcher Attribute von Tracks verglichen werden, und dann per Ähnlichkeitsanalyse entschieden werden kann, welche Tracks in eine Trackliste passen, um diese zu vervollständigen.

1.2 Collaborative Filtering

Collaborative-Filtering basiert auf der Annahme, dass Ähnlichkeiten zwischen den Interessen von Nutzern (hier Playlists) und Produkten (hier Tracks) existiert :

→ User-Based um Interessen des Users für das Filtering zu verwenden

→ Item-Based um Item-Ähnlichkeit (also hier Track-Ähnlichkeit in Playlists) für das Filtering zu verwenden

1.3 Filtern über Trackinfos

Infos über die Tracks werden über die Spotify API aus den TrackURIs gezogen Es wird kategorisiert und so neu gegliedert - so werden sinnvoll ähnliche Tracks zusammen gruppiert Hierzu zählen beispielsweise Genres wie Zugehörigkeit zu Rock oder Pop, und auch die Danceability (aussagekräftiger als bspw BPM, da BPM auch für die Danceability hinzugezogen wird). Mehr hierzu in 4.1.

1.4 Assoziationsanalyse - Unser Hauptansatz

Hier basierend auf dem Apriori Algorithmus. Umgesetzt mit der mlxtend-library und später ergänzt mit dem Frequent Pattern (FP) Growth Algorithmus, der ähnlich, aber für größere Datenmengen schneller, abläuft. [1]

Um die Daten richtig verwenden, und nach dem Apriori Algorithmus gescheit widerspiegeln zu können, müssen diese zunächst sanitized werden. [2]

1. Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.
2. How to use the apriori Algorithm for market basket analysis – Matt Clark

Zunächst erstellen wir also ein neues DataFrame namens baskets , in welche wir die Spalten, die wir brauchen zuweisen. In unserem Fall sind das playlist_id und track_uri. Nun verwenden wir die Pandas groupby() Funktion um auf beiden Spalten zu gruppieren. Letztlich werden noch mit sum die Quantitäten-Spalte berechnet, mit unstack die Hierarchie nach rechts verschoben und mit fillna(0) alle nan Werte mit 0 gefüllt.

```
df2 = df.assign(test=itemQuanity(df['track_uri']))
baskets = df2.groupby(['playlist_id', 'track_uri'])['test'].sum().unstack
                                        ().reset_index().fillna(0).set_index(
                                        'playlist_id')
baskets = baskets.applymap(encode_units)
```

Mit applymap(encode_units) werden die Werte auf 1 oder 0 encoded.

```
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
```

Als wir aber mit größeren Daten gearbeitet haben, gab es sehr lange Laufzeiten, da das Erstellen der Baskets sehr aufwändig war. Bei der Suche nach Alternativen haben wir letztlich den Transencoder von mlxtend entdeckt. [3]

Der TransactionEncoder ermöglicht uns ein Dataframe aus einer Liste, welche für alle Playlisten Listen mit Tracks enhalten, zu erstellen. Die Liste müsste dann diese Form :

```
[[track00, .. , track0N], ... ,[trackN0, .. , trackNN]]
```

```
te = TransactionEncoder()
te_data = te.fit(basket).transform(basket)
dfbasket = pd.DataFrame(te_data, columns=te.columns_)
```

Durch die Anwendung des TransactionEncoder mit der fit und der transform Methode wird eine 2D Liste erstellt, in der jeder innere liste die Information enthält, ob diese mindestens einen Träck enthält oder nicht. Mit Hilfe von Pandas erstellen wir dann ein Dataframe.
Jetzt kriegen wir per unserer (most recent) Apriori Implementation [4] [5] :

```
itemsets= fpgrowth(basket, use_colnames=True, verbose=0, min_support=0.001
                                        , max_len=3)
rules= association_rules(itemsets, metric="lift", min_threshold=1)
rules= rules.sort_values(['confidence', 'lift'], ascending=[False, False])
```

Regeln mit sowohl Konfidenz-Werten als auch Lift-Werten. Konfidenz-Werte besagen wie "sicher" sich der Algorithmus ist, dass zwei Songs am besten zusammen vorkommen sollten (bzw. eben zusammen vorkommen). Per diesem Wert sind Songs dann aufeinander (oder an Playlisten) anhängbar.

3. Mlxtend transaction encoder
4. Frequent Itemsets via the FP-Growth Algorithm
5. Frequent itemset via Apriori - Alter Ansatz

2 Vorgehensweise

Vorgehensweise zum erhalten der Daten, welche für das Ergebnis (submission) nötig ist.

2.1 Datenextrahierung

- DataAnalysis\loadSpotifyData -

Zunächst müssen alle relevanten Daten von allen Playlists die relevant sind aus der Datenbank extrahiert werden.
Somit müssen die :

— Playlist_id
— Anzahl der Tracks
— Kollektion von Tracks in einer Playlist
— Position von Tracks in dieser Kollektion

per Queries in eine extra CSV Datei (hier die test.csv Kaskade) übertragen werden.

2.2 Assoziationsregeln erstellen

- DataAnalysis\associationRuleMining -

Damit nun aus den in den vorherigem Schritt extrahierten Daten per Apriori Assoziationsregeln erstellt werden können, müssen die Daten in eine Datenstruktur umfunktioniert werden (aka. Pandas Dataframe). Mehr hierzu in 1.4.

2.3 Assoziationsregeln anwenden und Submission erstellen

- DataAnalysis\submissions -

Um nun letztlich die Submission File erstellen zu können, müssen die Regeln noch angewendet werden. Per Angabe der Aufgabenstellung in der Challenge, sollten pro Playlist insgesamt genau 500 neue Tracks vorkommen. Auch Dopplungen von Songs sind nicht erwünscht. Nun wird also für jeden Track in jeder Playlist überprüft, ob ein Track vorkommt, welcher einen anderen Track per Assoziationsregel "zugewiesen bekommen hat". So füllen wir die Playlists dementsprechend auf.

3 Unsere Submissions

3.1 Erster Apriori Ansatz

Bei unserem ersten Apriori Ansatz, haben wir unsere Baskets mit denen wir die Regeln erstellt haben aufgeteilt, um Multi-Processing zu ermöglichen. Dies ist allerdings Kontraproduktiv, da wir somit die einzelnen Samples nicht mehr mit allen Transaktionen vergleichen können. Es kamen also im Vergleich zum zweiten Ansatz viel wenigere Regeln dabei Raus. Dies spiegelt sich auch in der Bewertung unserer Submissions wieder.

175468	justisa	graded	0.064	0.096	13.385	Graded successfully!

3.2 Zweiter FP-Growth Ansatz

Da die Performance bei unserem zweiten Ansatz um einiges Akzeptabler war, war es uns Möglich den Algorithmus auf alle Testdaten gleichzeitig laufen zu lassen. Dies hat zu einer verdreifachung unseres Scores geführt.

ID	Participant	Status	R-prec	NDCG	Recommended Song Clicks	Message
175754	justisa	graded	0.135	0.220	8.707	Graded successfully!

4 Weiterführende Konzepte

4.1 Spotify-Api

Unsere Submission wird immernoch mit einem relativ niedrigen Score bewertet. Dies liegt vor allem daran, das wir bei leeren Playlists random Tracks einführen. Um dies zu verbessern haben wir den Ansatz verfolgt Metadaten von einzelnen Tracks auszulesen.

Infos über die Tracks werden über die Spotify API aus den TrackURIs gezogen. Es wird kategorisiert und so neu gegliedert - so werden sinnvoll ähnliche Tracks zusammen gruppiert. Hierzu zählen beispielsweise Genres wie Zugehörigkeit zu Rock oder Pop, und auch die Danceability (aussagekräftiger als bspw BPM, da BPM auch für die Danceability hinzugezogen wird). Diesen Ansatz könnte man über das Content based Filtering noch weiter verfolgen. Hierzu würden die Features zugezogen, und dann Playlists anhand dessen aufgestellt werden. Das hat zeitlich allerdings nicht ganz gereicht.

Leere Playlists mit beispielsweise dem Namen "Country" oder "Techno", würden dann mit Songs aufgefüllt werden, denen man diese Genres zugewiesen hat.