HAW HAMBURG

BACHELOR THESIS
Thu Vu Anh

# Vergleich von Reinforcement Learning und klassischen ML-Methoden für Musikempfehlungssysteme

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Thu Vu Anh

# Vergleich von Reinforcement Learning und klassischen ML-Methoden für Musikempfehlungssysteme

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 18.09.2025

**Thu Vu Anh**

**Thema der Arbeit**

Vergleich von Reinforcement Learning und klassischen ML-Methoden für Musikempfehlungssysteme

**Stichworte**

Empfehlungssyteme, RL, Data Science

**Kurzzusammenfassung**

In dieser Arbeit wird die Leistungsfähigkeit eines Reinforcement Learning (RL)-basierten Musikempfehlungssystems, implementiert mittels Deep Q-Learning (DQL), mit der klassischen Methode k-Nearest Neighbours (k-NN) verglichen. Ziel ist es, zu analysieren, unter welchen Bedingungen DQL Empfehlungen gegenüber k-NN Vorteile bieten und in welchen Szenarien k-NN überlegen bleibt.

Die Untersuchung erfolgt anhand synthetischer Nutzerprofile und realer Songdaten in einer selbstimplementierten Simulationsumgebung. Dabei werden zwei Arten von Musikentdeckung betrachtet: Vertiefung der bekannten Präferenzen und das Explorieren einer neuer Musikrichtung. Die Ergebnisse zeigen, dass k-NN insbesondere beim Auffinden ähnlicher Songs eine konsistente Nutzererfahrung liefert, während DQN seine Stärken vor allem bei explorativen Empfehlen von Musikrichtungen zeigt.

**Title of Thesis**

A Comparative Study of Reinforcement Learning and Classic ML-Methods for Music Recommendations Systems

**Keywords**

Recommender Systems, RL, Data Science

**Abstract**

In this work, the performance of a reinforcement learning (RL)-based music recommendation system, implemented using Deep Q-Learning (DQL), is compared with the classical k-Nearest Neighbours (k-NN) method. The goal is to analyze under which conditions DQL offer advantages over k-NN and in which scenarios k-NN remains superior.

The investigation is conducted using synthetic user profiles and real song data within a self-implemented simulation environment. Two types of music discovery are considered: deepening known preferences and exploring new music genres. The results show that k-NN provides a consistent user experience, particularly when identifying similar songs, while DQN demonstrates its strengths primarily in exploratory recommendations across different music genres.

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **vii**

**Tabellenverzeichnis** **viii**

**Abkürzungen** **ix**

**1 Einleitung** **1**
1.1 Motivation . . . 1
1.2 Verwandte Arbeiten . . . 2
1.3 Ziel der Arbeit . . . 3
1.4 Aufbau der Arbeit . . . 3

**2 Grundlagen** **4**
2.1 Empfehlungssystem . . . 4
2.1.1 Datensammlung . . . 5
2.1.2 Empfehlungstechniken . . . 7
2.1.3 Musikempfehlung . . . 10
2.2 Klassische Machine Learning-Methoden . . . 10
2.2.1 K-Nearest Neighbour . . . 11
2.3 Reinforcement Learning . . . 12
2.3.1 Q-Learning . . . 13
2.3.2 Deep Q-Learning . . . 15

**3 Methodik und Experimentaufbau** **17**
3.1 Methodik . . . 17
3.1.1 Auswahl der Algorithmen . . . 18
3.1.2 Datengrundlage . . . 19
3.1.3 Simulationsumgebung . . . 22
3.1.4 Modellierung der Empfehlungssysteme . . . 27

3.2 Experimentaufbau . . . . . 31
3.2.1 Durchführung der Experimente . . . . . 32
3.2.2 Auswahl der Evaluationsmetriken . . . . . 33

**4 Ergebnisse** **35**
4.1 Ergebnisse der Experimente per Modell . . . . . 35
4.1.1 Genre-gebundenen Zufallsauswahl (Random) . . . . . 35
4.1.2 k-Nearest Neighbours (k-NN) . . . . . 36
4.1.3 Deep Q-Network (DQN) . . . . . 38
4.2 Ergebnisse im Vergleich . . . . . 41
4.2.1 Präzision . . . . . 41
4.2.2 Diversität . . . . . 42
4.2.3 Stabilität . . . . . 43
4.2.4 Empfehlungsqualität . . . . . 44

**5 Diskussion** **46**
5.1 Interpretation der Ergebnisse . . . . . 46
5.2 Praktische Relevanz . . . . . 48
5.3 Limitationen . . . . . 48
5.4 Ausblick . . . . . 50

**Literaturverzeichnis** **51**

**A Anhang** **53**
A.1 Verwendete Hilfsmittel . . . . . 58

**Selbstständigkeitserklärung** **59**

# Abbildungsverzeichnis


2.1 User-based Collaborative Filtering . . . . . 8
2.2 Content-based Filtering . . . . . 8
2.3 Ablauf eines Reinforcement-Learning-Agents . . . . . 12

3.1 Visualisierung der Song-Embeddings . . . . . 20
3.2 Verteilung der Song-Ähnlichkeiten zu einem Referenzsong . . . . . 24
3.3 Trainingsverlauf DQN . . . . . 30
3.4 Verteilung der Nutzerreaktionen im Trainingsverlauf . . . . . 31

4.1 Verteilung der Nutzerreaktionen k-NN . . . . . 37
4.2 Verteilung der Nutzerreaktionen DQN . . . . . 39
4.3 Vergleich der Präzision für korrekte Profile . . . . . 41
4.4 Vergleich der Präzision für inkorrekte Profile . . . . . 41
4.5 Vergleich der Diversität für korrekte Profile . . . . . 42
4.6 Vergleich der Diversität für inkorrekte Profile . . . . . 42
4.7 Vergleich der Streak-Länge für korrekte Profile . . . . . 43
4.8 Vergleich der Streak-Länge für inkorrekte Profile . . . . . 43
4.9 Vergleich der Reaktionsanteile für korrekte Profile . . . . . 44
4.10 Vergleich der Reaktionsanteile für inkorrekte Profile . . . . . 44

A.1 t-SNE Visualisierung der Features von 1200 Songs: Vergleich Librosa vs. OpenL3 . . . . . 54
A.2 t-SNE Visualisierung der OpenL3 Embeddings pro Genre . . . . . 55
A.3 Available songs per user (correct) . . . . . 56
A.4 Available songs per user (incorrect) . . . . . 56
A.5 Übersicht der Testnutzer mit ihrer möglichen Anzahl an positiven Empfehlungen, sortiert nach Genre . . . . . 56
A.6 Verteilung der Nutzerreaktionen genre-gebundenen Zufallsauswahl . . . . 57

# Tabellenverzeichnis


2.1 User–Item Interaktionsmatrix . . . . . 7

3.1 Übersicht der Metadaten der Songs im Datensatz . . . . . 19
3.2 Intra-Genre-Similarity Werte . . . . . 21
3.3 Durchschnittliche Anzahl unähnlicher, moderat ähnlicher und stark ähnlicher Songs pro Song im Genre . . . . . 21
3.4 Beispiele für extrem ähnliche und unähnliche Songs innerhalb eines Genres 22
3.5 Mögliche Nutzerreaktionen auf Songs . . . . . 26
3.6 Mittelwerte der Anzahl an relevanten Songs pro Genre . . . . . 33

4.1 Zufallsauswahl: Korrekte Profile . . . . . 35
4.2 Zufallsauswahl: Inkorrekte Profile . . . . . 36
4.3 k-NN: Korrekte Profile . . . . . 36
4.4 k-NN: Inkorrekte Profile . . . . . 37
4.5 DQN: Korrekte Profile . . . . . 38
4.6 DQN: Inkorrekte Profile . . . . . 38

A.1 Jahresstatistik der Songs pro Genre . . . . . 53
A.2 Längenstatistik der Songs pro Genre . . . . . 53
A.3 Popularitätsstatistik der Songs und Anzahl an distinkte Künstler pro Genre . . . . . 53
A.4 Verwendete Hilfsmittel und Werkzeuge . . . . . 58

# Abkürzungen

**CBF** Content-based Filtering.

**CF** Collaborative Filtering.

**DQL** Deep Q-Learning.

**DQN** Deep Q-Network.

**k-NN** k-Nearest Neighbours.

**MDP** Markov-Entscheidungsprozesse.

**ML** Machine Learning.

**PER** Prioritized Experience Replay.

**RL** Reinforcement Learning.

# 1 Einleitung

## 1.1 Motivation

Mit dem stetig wachsenden Musikangebot auf Streaming-Plattformen wird es immer schwieriger, neue und passende Musik zu entdecken. Kollaborative Empfehlungssysteme, die auf den Vorlieben und dem Verhalten anderer Nutzer basieren, bevorzugen oft populäre Inhalte [1]. Obwohl diese Systeme in Bereichen wie Filmen oder Büchern gut funktionieren, gestaltet sich die Erkennung des individuellen Musikgeschmacks schwieriger, da dieser durch oft schwer fassbare Faktoren beeinflusst wird. Zudem führt dies häufig zu einer Wiederholung von Empfehlungen, die wenig Abwechslung bieten, etwa wenn nach dem Hören eines bestimmten Liedes von einem Künstler immer wieder nur Lieder desselben Künstlers vorgeschlagen werden.

Content-basierte Empfehlungssysteme, die Musik anhand ihrer musikalischen Eigenschaften wie Tempo, Rhythmus und Instrumentierung empfehlen, bieten eine Lösung, um eine größere Vielfalt in den Empfehlungen zu erreichen. Diese Systeme ermöglichen es, Musik zu empfehlen, die nicht nur ähnliche Vorlieben trifft, sondern auch neue und weniger populäre Künstler und Lieder zu entdecken.

Es gibt unterschiedliche Ansätze, um Empfehlungen in solchen Systemen zu erzeugen. Traditionelle Machine-Learning-Methoden liefern oft gute Ergebnisse, sind jedoch in ihrer Anpassungsfähigkeit eingeschränkt und konzentrieren sich primär auf die Vertiefung bestehender Präferenzen. Die explorative Entdeckung neuer Inhalte bleibt dabei häufig unberücksichtigt.

Reinforcement Learning (RL) eröffnet hier neue Möglichkeiten, da es das Nutzerverhalten dynamisch berücksichtigt und kontinuierlich daraus lernt. Dadurch kann RL potenziell eine bessere Balance zwischen der Vertiefung bekannter Vorlieben und der Förderung explorativer Musikentdeckung erreichen.

Die vorliegende Arbeit untersucht den Vergleich zwischen einem klassischen Machine-Learning-Ansatz, **k-NN**, und einem RL-basierten Ansatz, implementiert mittels **Deep Q-Learning (DQL)**. Analysiert wird, unter welchen Bedingungen RL gegenüber k-NN Vorteile bietet und in welchen Szenarien klassische Methoden weiterhin überlegen sind.

Der Vergleich erfolgt auf Basis synthetischer Nutzerprofile und realer Songdaten in einer selbst entwickelten Simulationsumgebung. Dabei wird sowohl die Vertiefung bekannter Präferenzen als auch die explorative Musikentdeckung betrachtet, um die Leistungsfähigkeit der Ansätze in unterschiedlichen Anwendungsszenarien zu evaluieren.

## 1.2 Verwandte Arbeiten

Die Forschung im Bereich musikbasierter Empfehlungssysteme deckt sowohl erweiterte Machine Learning (ML) Methoden als auch RL Ansätze ab. Zahlreichende Arbeiten zeigen, dass RL-Systeme besonders geeignet sind, Nutzerinteraktionen dynamisch zu berücksichtigen und langfristige Belohnung zu optimieren.

Mok et al. [2] liefern empirische Einblicke in Nutzerverhalten auf Spotify und verdeutlichen, dass Musik sowohl durch Vertiefung bekannter Präferenzen als auch durch explorative Entdeckung konsumiert wird.

Tomas et al. [3] untersuchen die automatische Playlist-Generierung mittels Simulationsbasierter RL-Methoden. Sie nutzen eine komplexe Umgebung, in der der Agent kontinuierlich aus Nutzerfeedback lernt, um personalisierte Playlists zu erstellen.

Liebman et al. [4] stellen ein neuartiges Reinforcement-Learning-Framework DJ-MC für Musikempfehlung vor. Anderes als klassische Systeme empfiehlt DJ-MC ganze Songsequenzen und nutzt dafür ein Modell für Präferenzen einzelner Songs als auch für Übergänge zwischen Songs. Auch soll das Modell online lernen und individuell für jeden Hörer angepasst werden.

Chen et al. [5] schlagen ein Modell für sequentielle Empfehlungen vor, das das Standardmodell durch einen zusätzlichen Output-Layer erweitert. Dadurch kann ein selbst-überwachtes Netzwerk mittels Reinforcement Learning gezielt auf bestimmte Rewards ausgerichtet werden.

## 1.3 Ziel der Arbeit

Ziel dieser Arbeit ist es, klassische ML-basierte und RL-basierte Empfehlungssysteme für Musikentdeckung zu implementieren und vergleichend zu evaluieren.

Untersucht wird, in welchem Maß die Systeme individuelle Präferenzen korrekt abbilden und flexibel auf Veränderungen im Nutzerverhalten reagieren. Dabei werden zwei Nutzerverhalten modelliert: Nutzer, die ihre Präferenzen vertiefen und Nutzer, die ihre Präferenzen erweitern.

Im Rahmen dieser Arbeit werden folgende Schritte durchgeführt:

- Etablierung der Datengrundlage: Auswahl und Aufbereitung eines Songdatensatzes inklusive relevanter Metadaten und Audiofeatures
- Entwicklung einer Simulationsumgebung: Modellierung des Nutzerverhaltens und Definition der Interaktionslogik zwischen Nutzer und Empfehlungssystem
- Implementierung zweier Empfehlungssysteme:

  k-NN basierter Ansatz für content-based Musikempfehlung

  DQL basierter Ansatz zur schrittweisen Verbesserung von Musikempfehlung
- Definition von Evaluationsmetriken: zur Bestimmung geeigneter Kennzahlen zur Messung der Empfehlungsqualität im Hinblick auf Musikentdeckung
- Experimentelle Evaluation: Vergleich beider Ansätze auf definierte Metriken und Analyse der Ergebnisse

## 1.4 Aufbau der Arbeit

Die Arbeit ist wie folgt aufgebaut: Kapitel 2 behandelt die theoretischen Grundlagen von Empfehlungssystemen, Musikempfehlung und Reinforcement Learning. Kapitel 3 beschreibt Methodik und Experimentaufbau, inklusive Datengrundlage, Simulationsumgebung und Umsetzung der Systeme. Kapitel 4 präsentiert die Evaluation und den Vergleich der Modelle. Kapitel 5 fasst die Ergebnisse zusammen, diskutiert Limitationen und zeigt Ansätze für zukünftige Arbeiten auf.

# 2 Grundlagen

Dieses Kapitel stellt die theoretischen Grundlagen für die Entwicklung eines Musikempfehlungssystems vor. Es behandelt zunächst die Funktionsweise und Aufgaben von Empfehlungssystemen, gefolgt von einer Diskussion Machine-Learning-Verfahren und Reinforcement Learning. Abschließend werden spezifische Herausforderungen bei der Musikempfehlung sowie relevante Datengrundlagen erläutert.

## 2.1 Empfehlungssystem

Empfehlungssysteme sind Softwaresysteme, die Nutzern helfen, in großen Informationsmengen relevante Inhalte zu finden. Der Begriff „Item“ bezeichnet dabei allgemein das Objekt, das empfohlen wird, wie z. B. ein Produkt, ein Lied oder ein Film. Die Architektur und Algorithmen eines Empfehlungssystems werden an die spezifischen Eigenschaften der Items und den Anwendungskontext angepasst, um möglichst passende Vorschläge zu machen. Typische Anwendungsfelder sind E-Commerce (Produktempfehlungen), Video- und Musikstreaming (z. B. Filme, Serien, Songs), Nachrichtenportale oder soziale Netzwerke. Die Systeme unterstützen Nutzer bei der Entscheidungsfindung, indem sie Items vorschlagen, die für einen Nutzer in einem bestimmten Kontext potenziell von Interesse sind. [6, S.1]

Funktional lassen sich die meisten Empfehlungssysteme in verschiedene Aufgaben einteilen. Gunawardana und Shani [7, S.2938-2940] unterscheiden drei grundlegende Aufgaben von Empfehlungssystemen: Recommend Good Items, Prediction und Utility Optimization.

**Prediction** umfasst die Schätzung der Präferenzen eines Nutzers für ein Item, typischerweise durch die Vorhersage einer Bewertung (Rating) z. B. eine Bewertung aus eins bis fünf Sterne. Solche Systeme erzeugen keine Empfehlungsliste im klassischen Sinne, dennoch kann eine hohe vorhersagte Bewertung als Empfehlung zum Konsum und eine niedrige Bewertung als Empfehlung zur Vermeidung interpretiert werden.

**Utility Optimization** betrachtet Empfehlungen als Handlung, die eine bestimmte Nutzenfunktion maximiert. Diese Funktion ordnet jede Empfehlung einen Wert zu, der den erwarteten Nutzen für den Nutzer oder System ausdrückt. Die genaue Definition der richtigen Nutzenfunktion für eine Anwendung ist jedoch oft schwierig. Beispielsweise könnte ein E-Commerce-System die Wahrscheinlichkeit eines Kaufs maximieren.

Bei **Recommend Good Items** wird eine Menge relevanter Items vorgeschlagen, die für die Nutzer von hohem Interesse sind. Dabei wird angenommen, dass die vorgeschlagenen Items ähnliche Relevanz besitzen und in einer ungeordneten Liste präsentiert werden können. Besonders wenn die Menge an relevante Items zu groß ist, aber der Nutzer nicht die Ressourcen (Geld, Zeit) hat, um alle Items auszuwählen. In Medienplattformen wie Youtube oder Instagram ist die Menge an verfügbaren Inhalten so groß, dass der Nutzer nicht genug Zeit hat alle potenziell relevanten Items zu konsumieren. Daher liegt der Fokus darauf, irrelevante Inhalte zu vermeiden, anstatt sämtliche relevante Items vorzuschlagen.

### 2.1.1 Datensammlung

Die Qualität und Verfügbarkeit der Daten sind entscheidend für die Leistungsfähigkeit von Empfehlungssystemen. Sie ist die Datengrundlage auf der die Empfehlungen beruhen. Nur durch die Erfassung relevanter Informationen über Nutzer, Items und deren Interaktionen können Muster erkannt und Präferenzen modelliert werden. Dabei werden unterschiedliche Datentypen erfasst, darunter Item-Daten (z. B. Metadaten, Audiofeatures), Nutzerdaten (z. B. Bewertungen) sowie Interaktionsdaten (z. B. Wiedergabenlisten).

Viele dieser Informationen liegen in unstrukturierter Form vor, beispielsweise als Text, Audiodateien oder Log-Daten. Damit sie für Algorithmen nutzbar sind, müssen sie zunächst bereinigt, gefiltert und in einem interpretierbaren Format überführt werden. Dazu gehören Schritte wie:

- Datenbereinigung: Entfernen fehlerhafter, redundanter oder unvollständiger Daten
- Normalisierung: Vereinheitlichung von Skalen und Formaten (z. B. Bewertungen)
- Feature-Extraktion: Ableitung relevanter Merkmale aus Rohdaten, etwa Audio oder Text

- Numerische Transformation: Umwandlung in Vektorrepräsentationen, die für Modelle geeignet sind

### Data Sparsity

Ein zentrales Problem ist Data Sparsity (Datenknappheit), bei der nur ein kleiner Teil der möglichen Nutzer-Item-Interaktionen bekannt ist. Diese kleine Menge an beobachteten Interaktionen ist nicht aussagekräftig genug, um Präferenzen zu interpretieren, wodurch die Qualität der Empfehlungen negativ beeinflusst werden.

Eng damit verbunden ist das Cold-Start-Problem, das auftritt, wenn ein Empfehlungssystem für neue Nutzern oder neue Items noch keine ausreichenden Daten hat, um gute Empfehlungen zu generieren [6, S.13]. Ohne historische Interaktionen fehlen wichtige Informationen zur Personalisierung. Für neue Nutzern ist das kein Problem, weil Systeme erstmal beliebte oder allgemein gut bewertete Items empfehlen bis genug Daten vorliegen, um den Inhalt zu personalisieren. Neue Items hingegen bleiben für Nutzer so lange unsichtbar bis sie genug Bewertungen haben.

In Systemen mit einer großen Anzahl an bestehenden Items ist die Wahrscheinlichkeit gering, dass neue Items ohne aktive Interaktionen entdeckt und bewertet werden. Das Cold-Start-Problem erschwert besonders neuen Inhalten, Sichtbarkeit zu erlangen und sich im Empfehlungssystem durchzusetzen, was ihre Verbreitung und Bewertung stark einschränkt.

### Item-Embeddings

Um das beschriebene Problem von Data Sparsity zu reduzieren, werden häufig Item-Embeddings verwendet. Item Embeddings sind niedrig dimensionale Vektordarstellungen von Items, die semantische ähnliche Items so kodieren, dass sie im Raum nah beieinander liegen. Durch Projektion in einen kontinuierlichen Vektorraum können Empfehlungssysteme Beziehungen zwischen Items erfassen, die über die beobachteten Interaktionen hinausgehen. Die Erstellung von Item-Embeddings kann auf verschiedenen Informationsquellen basieren, oft über Metadaten, inhaltliche Features oder Ko-Interaktionen aus Nutzer-Item Matrizen.

Ein Vorteil der Embeddings ist, dass sie sowohl für klassische Empfehlungssysteme als auch für maschinelle Lernverfahren wie neuronale Netze genutzt werden können. Sie

helfen die Probleme der Data Sparsity zu mildern, indem sie fehlende Interaktionen durch Ähnlichkeitsinformationen zwischen Items teilweise kompensieren.

Die spezifische Datenbasis und deren Verarbeitung werden im Methodikteil erläutert.

### 2.1.2 Empfehlungstechniken

Um relevante Inhalte vorzuschlagen, müssen Empfehlungssysteme den potenziellen Nutzen eines Items für einen Nutzer einschätzen. Diese Abschätzung kann explizit über Bewertungen (Ratings) oder implizit über Heuristiken erfolgen. Implizite Bewertungen basieren auf Nutzerverhalten wie z. B. Klicks, Wiedergabezeiten oder Überspringen von Inhalten, während explizite Bewertungen direkt durch Nutzer angegebene Ratings (z. B. Like/Dislike) repräsentiert werden. Auf Basis dieser Bewertung wählt das System dann die besten Items zur Empfehlung aus.

Zwei der am weitesten verbreiteten Techniken sind Collaborative Filtering und Content-Based Filtering.

#### Collaborative Filtering

Bei Collaborative Filtering (CF) werden Interaktionen zwischen Nutzern und Items, z. B. Bewertungen, in einer Matrix dargestellt (siehe Tabelle 2.1). Die Aufgabe ist dann, die

| | Item 1 | Item 2 | Item 3 |
|---|---|---|---|
| User A | 4 | ? | 4 |
| User B | 4 | 2 | ? |
| User C | ? | 5 | 4 |

Tabelle 2.1: User–Item Interaktionsmatrix

fehlenden Bewertungen und Items vorherzusagen und dem Nutzer, die noch unbekannten Items mit den höchsten Vorhersagen zu empfehlen. Man unterscheidet hier zwischen user-based CF und item-based CF, je nachdem, ob die Empfehlungen auf Ähnlichkeit zwischen Nutzern oder zwischen Items basiert. Beim user-based CF werden Nutzer mit ähnlichen Präferenzen identifiziert, und Items, die von diesen Nutzern positiv bewertet wurden, werden empfohlen (siehe Abbildung 2.1).

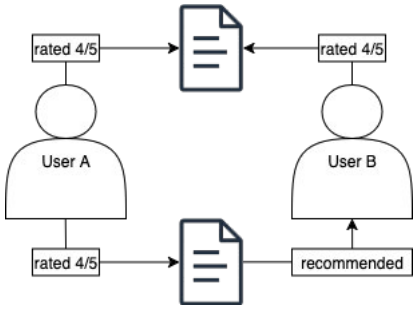



Abbildung 2.1: User-based Collaborative Filtering

Beim item-based CF werden Items basierend auf ihrer Ähnlichkeit in den Bewertungen von Nutzer vorgeschlagen.

Der Vorteil von CF ist, dass extensive Analyse der Items benötigt, die sie empfehlen. Stattdessen basieren die Empfehlungen allein auf dem Verhalten der Nutzer, was besonders dann hilfreich ist, wenn keine oder nur wenige Item-Metadaten vorliegen.

Der Nachteil von CF ist, dass es anfällig für das Cold-Start-Problem ist, da dieser sehr auf Bewertungen abhängig ist.

### Content-based Filtering

Bei Content-based Filtering (CBF) werden Empfehlungen auf Basis der Merkmale der Items erstellt. Ziel ist es, Items zu empfehlen, die inhaltlich ähnlich zu denjenigen sind, die der Nutzer bereits positiv bewertet hat (siehe Abbildung 2.2).

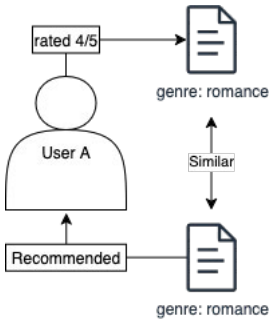



Abbildung 2.2: Content-based Filtering

Die Ähnlichkeit wird durch den Vergleich von Features ermittelt, die aus den Items extrahiert wurden, z. B. Audioeigenschaften bei Musik. Dazu wird für jedes Item ein

Merkmalsvektor (Item-Embedding) erstellt, die die relevanten Merkmale abbildet. Ein Nutzerprofil wird durch eine gewichtete Darstellung der Merkmale erstellt, die auf den bisherigen Präferenzen des Nutzers basieren. Die Empfehlungen erfolgen dann, indem neue Items mit dem Nutzerprofil verglichen und ähnliche Items vorgeschlagen werden [8, S. 6].

Cosine-Similarity [9, S. 3] ist ein bekanntest Maß, um die Ähnlichkeit zwischen zwei nicht-null Vektoren in einem $n$-dimensionalen Raum zu erfassen. Mit der Formel 2.1 wird der Winkel zwischen den zwei Vektoren berechnet, um ihre Orientierung zu bestimmen.

$$\cos \alpha = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}|\,|\mathbf{B}|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\,\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.1}$$

Diese Ähnlichkeitsmetrik hat einen Wertebereich von $-1$ bis 1, die folgend interpretiert werden:

- -1: Vektoren stehen gegenüber
- 0: Vektoren sind orthogonal
- 1: Vektoren sind gleich

Der Vorteil von CBF ist, dass neue Items sofort empfohlen werden können, sobald ihre Merkmale bekannt sind. Das System ist somit nicht auf frühere Nutzerinteraktionen angewiesen und eignet sich gut zur Bewältigung des Cold-Start-Problems für neue Items.

Jedoch muss mehr Aufwand in die Feature-Extraktion gesetzt werden, um die Merkmale der Items zu sammeln. Oft ist es schwierig, die richtigen Merkmale zu wählen, die den Inhalt sinnvoll und nutzerrelevant beschreiben. Außerdem erfordert die Extraktion, besonders für Musik, spezialisierte Verfahren und Rechenressourcen, was den Einsatz in der Praxis erschwert. Zwar gibt es inzwischen Machine-Learning-Modelle, die diese Arbeit automatisieren und vereinfachen, jedoch bleibt die Auswahl und Interpretation der Features eine zentrale Herausforderung.

Ein weiterer Nachteil ist, dass die Vielfalt der Empfehlungen eingeschränkt wird, Da inhaltlich nur ähnliche Items empfohlen werden, entsteht eine selbstverstärkende Schleife. Die Folge darauf ist, dass der Nutzer keine neuen oder unerwartete Inhalte entdeckt und im Extremfall in dieser begrenzten Auswahl von Inhalten gefangen ist. Beispielsweise könnte ein Nutzer, der hauptsächlich nur Popmusik hört, nur ähnliche Pop empfohlen bekommen, während andere Genres wie Rock oder Hip-Hop unterrepräsentiert bleiben.

Zudem können die Songs so ähnlich zueinander sein, dass der Nutzer an den Empfehlungen Interesse verliert.

### 2.1.3 Musikempfehlung

Musikempfehlungen unterscheiden sich von anderen Bereichen wie Filmen oder Büchern und stellen spezifische Anforderungen an Empfehlungssysteme [10, S. 3-4].

Zu einem ist die kurze Dauer eines Songs, diese sind im Durchschnitt drei bis vier Minuten lang. Kombiniert mit der Millionen Anzahl an Songs, die online zugängig sind, ist Musik heutzutage deutlich austauschbarer als andere Inhalte, wie zum Beispiel Möbeln. Fehlentscheidungen fallen daher weniger ins Gewicht, da Nutzer schnell überspringen können, wenn es ihnen nicht gefällt. Bei Filmen dauert es deutlich länger bis der Nutzer merkt, dass ihnen der Film nicht gefällt, weshalb sie mehr enttäuscht sind über eine schlechte Empfehlung.

Wiederholtes Hören wird dabei meist positiv bewertet, da Musik eher passiv konsumiert wird. Erste Eindrücke können sich deshalb erst nach mehreren Durchläufen vollständig bilden.

Ein Urteil über einen Song kann sich mit dem Kontext ändern. Zum Beispiel wird ein Nutzer beim Sport lieber energiegeladene Musik wie EDM, weil sie besser zur körperlichen Aktivität passt. Aber außerhalb vom Training zeigt der Nutzer eher Interesse an Rock-Songs und ist eher abgeneigt von EDM-Empfehlungen.

Oft wird Musik in Sequenzen abgespielt, weshalb Empfehlungssysteme entweder einzelne Songs oder zusammenhängende Playlists vorschlagen müssen. Bei Playlists kann der Nutzer die Empfehlungen vorab einsehen und entscheiden, welche Songs er hört. Einzelne Song-Empfehlungen ermöglichen es hingegen, den Nutzer aktiv in den Empfehlungsprozess einzubeziehen und seine Reaktionen für weitere Entscheidungen zu berücksichtigen.

## 2.2 Klassische Machine Learning-Methoden

Klassische Machine-Learning-Verfahren sind etablierte Algorithmen zur Mustererkennung, die auf statischen Trainingsdaten basieren. Sie arbeiten auf Basis statischer Datensätze und nutzen mathematische Modelle, um Muster zu erkennen oder Vorhersagen zu treffen.

Typischerweise unterscheidet man zwischen überwachtes (supervised) und unüberwachte (unsupervised) Verfahren.

Unter Supervised Learning gibt es zwei zentrale Ansätze, Klassifikation und Regression. Beide nutzen gelabelte Trainingsdaten, aber verfolgen unterschiedliche Ziele.

Bei der Klassifikation möchte man Eingaben in einer bestimmten Kategorie bzw. Klasse zuordnen. Zum Beispiel soll ein Modell vorhersagen, ob eine E-Mail „Spam“ oder „Nicht-Spam“ ist. Es wird mit Beispielen trainiert, die aus E-Mails und dem Labeln „Spam“/„Nicht-Spam“ bestehen, und lernt neue E-Mails einer der beiden Klassen zuzuordnen.

Die Regression hingegen sagt einen kontinuierlichen numerischen Wert voraus. Ein Beispiel wäre ein Modell, dass Wohnungspreise basierend auf Merkmalen wie Größe, Lage und Baujahr vorhersagt. Dieses Modell wird mit Beispieldaten bestehend aus Wohnungen und dessen tatsächlichen Preisen trainiert und gibt als Ausgabe den geschätzten Preis an.

Im Kontext von Empfehlungssystemen werden solche Verfahren eingesetzt, um Nutzerpräferenzen abzuleiten, Item-Ähnlichkeiten zu berechnen oder bewertungsbasierte Vorhersagen zu treffen.

### 2.2.1 K-Nearest Neighbour

K-Nearest Neighbours (k-NN) ist ein einfaches, aber effektives überwachtes Lernverfahren, das sowohl für Klassifikation als auch Regression eingesetzt werden kann. Es gehört zu den sogenannten Lazy-Learning-Algorithmen, da kein expliziter Trainingsschritt stattfindet. Stattdessen wird bei jeder Vorhersage die Ähnlichkeit eines neuen Datenpunkts zu bereits bekannten Beispielen bestimmt.

Das Grundprinzip besteht darin, für eine gegebene Eingabe die $k$ ähnlichsten Datenpunkte bzw. Nachbarn im Datensatz zu identifizieren. Die Ähnlichkeit wird dabei meist mittels einer Distanzfunktion wie euklidische Distanz oder Cosine Similarity 2.1, die die Merkmale der Datenpunkte verwendet. Bei der Klassifikation bestimmt k-NN die Zielklasse des neuen Beispiels basierend auf der Mehrheitsklasse unter den $k$-Nachbarn.

In Empfehlungssystemen kann k-NN dazu verwendet werden, ähnliche Nutzer oder Items zu finden. Für content-basierte Musikempfehlung lassen sich etwa Songs vorschlagen, die ähnliche Merkmale wie bereits vom Nutzer positiv bewerteten Songs haben.

## 2.3 Reinforcement Learning

Reinforcement Learning (RL) ist ein Teilgebiet von Machine Learning, der sich auf das Lernen durch Interaktionen mit einer Umgebung und Belohnungssignalen konzentriert. Die lernende Einheit, der Agent, wird muss durch Versuche herausfinden, welche Aktionen in welcher Situation Belohnungen geben. Bei komplexen Problemen können Aktionen auch alle zukünftigen Belohnungen beeinflussen, je nachdem welche Situation sich nach der Aktion ergibt.

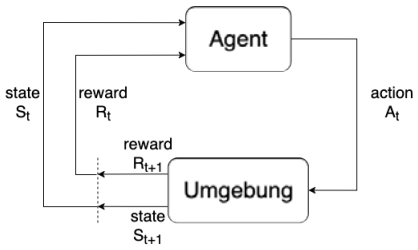



Abbildung 2.3: Ablauf eines Reinforcement-Learning-Agents

Dabei beobachtet der Agent den Zustand $S_t$ der Umgebung, wählt daraufhin Aktionen $A_t$ aus. Die Umgebung führt die Aktion des Agenten aus und schaltet in einen neuen Zustand $S_{t+1}$ und berechnet eine Belohnung $R_{t+1}$ für den Agenten. Die Belohnung $R_{t+1}$ wird dem Agenten dazu verleiten seinen Verhalten anzupassen und mit dem neuen Zustand $S_{t+1}$ wiederholt sich dieser Prozess iterativ bis das Ziel erreicht wurde oder eine Abbruchbedingung erfüllt ist. Bei diesem Lernprozess (siehe Abbildung 2.3) verbessert der Agent mit jedem Versuch seine Strategie.

RL-Probleme lassen sich als Markov-Entscheidungsprozesse (MDP) mit Zuständen, Aktionen, Übergangswahrscheinlichkeiten und Belohnungen formulieren [11, S. 2]. Eine Policy definiert das Verhalten des Agenten, während Value Functions den erwarteten zukünftigen Nutzen eines Zustands oder Aktion bewerteten. Ziel ist es eine Policy zu erlernen, die in verschiedenen Situationen möglichst hohe kumulierte Belohnungen erzielt.

Model-free Reinforcement Learning bezeichnet Verfahren, bei denen Agenten kein Modell der Umgebung lernen oder nutzen. Das bedeutet, der Agent trifft ausschließlich Entscheidungen basierend auf Erfahrungen ohne die Übergangsdynamik oder Belohnungsfunktion der Umgebung explizit zu kennen oder zu schätzen. Unter Erfahrung sind beobachteten Zustände, Aktionen und Belohnungen gemeint, die mit jeder Interaktion entstehen. Dies eignet sich für komplex oder unbekannte Umgebungen.

Der Agent muss einen Kompromiss zwischen Exploration (neue Aktionen ausprobieren) und Exploitation (bekannte, erfolgreiche Aktionen nutzen) finden. Nur so kann er optimale Entscheidungen lernen [11, S. 3]. Zum Beispiel wird ein Agent, der nur Lieder empfiehlt, die der Nutzer bereits kennt oder mag (Exploitation), keine neuen Entdeckungen bieten. Empfiehlt er hingegen gelegentlich unbekannte Songs aus ähnlichen oder völlig neuen Genres (Exploration), kann er die musikalischen Vorlieben des Nutzers erweitern oder verfehlen.

Eine gängige Umsetzung dieses Kompromisses ist der $\epsilon$-greedy-Strategie. Mit Wahrscheinlichkeit $\epsilon$ wählt der Agent zufällig eine Aktion (Exploration), ansonsten die beste bekannte Aktion (Exploitation). Anfangs ist $\epsilon$ hoch, um viele Aktionen auszuprobieren, und wird im Verlauf schrittweise reduziert, sodass der Agent zunehmend auf gelerntes Wissen zurückgreift.

### 2.3.1 Q-Learning

Q-Learning ist ein model-free Reinforcement-Learning-Verfahren, mit dem ein Agent lernt, wie gut bestimmte Aktionen in bestimmten Zuständen sind. Dazu schätzt der Agent eine Funktion $Q(s, a)$, die den erwarteten zukünftigen Nutzen bzw. Belohnung angibt, wenn er im Zustand $s$ die Aktion $a$ ausführt und danach optimal handelt. Ziel ist es, die optimale action-value $q^*(s, a)$ zu approximieren. Diese beschreibt den maximal möglichen Erwartungswert der kumulierten Belohnungen, wenn Zustand $s$ die Aktion $a$ wählt und danach optimal handelt.

Für jeden Zustand und jede Aktion werden die Q-Werte in einer Tabelle gespeichert. Mithilfe der Bellman-Gleichung ergibt sich die Aktualisierungsformel für ein Zustand-Aktions-Paar:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$$

**Dabei gilt:**

| | |
|---|---|
| $Q(s, a)$ | Geschätzter Wert der Aktion $a$ im Zustand $s$ (Q-Wert) |
| $\alpha$ | Lernrate, $0 < \alpha \leq 1$ |
| $r$ | Unmittelbare Belohnung nach Ausführen von $a$ in $s$ |
| $\gamma$ | Discountfactor für zukünftige Belohnungen, $0 \leq \gamma < 1$ |
| $\max_{a'} Q(s', a')$ | Maximaler geschätzter Wert im Folgezustand $s'$ über alle $a'$ |
| $s'$ | Folgezustand nach Ausführen von $a$ in $s$ |
| $a'$ | Mögliche Aktion im Folgezustand $s'$ |

Der optimale Q-Wert für $(s, a)$ ist der sofortige Reward $r$ und der diskontierte beste Q-Wert des nächsten Zustands. Da Q-Learning eine $\epsilon$-greedy Strategie verwendet, exploriert der Agent zu Beginn des Trainings häufig und wählt später zunehmend die aktuell beste bekannte Aktion. Auf dieser Weise lernt er mit jeder Interaktion, welche Aktionen langfristig den höchsten Nutzen bringen, ohne ein explizites Modell der Umgebung zu haben. Der Algorithmus 1 fasst den Ablauf von Q-Learning zusammen und zeigt, wie die Q-Werte mit der Bellman-Gleichung aktualisiert werden.

**Algorithm 1** Q-Learning

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$;
Initialize $Q(s, a)$ for all $s \in S, a \in A$ arbitrarily, except $Q(\text{terminal}, \cdot) = 0$;
Set learning rate $\alpha$, discount factor $\gamma$, and $\epsilon$
**for** each episode **do**
    Initialize state $s$
    **while** $s$ is not terminal **do**
        Choose action $a$ using $\epsilon$-greedy policy
        Take action $a$, observe reward $r$ and next state $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$
        $s \leftarrow s'$
    **end while**
**end for**

Des Weiteren ist Q-Learning ein off-policy-Verfahren und lernt den Wert der optimalen Aktionen unabhängig von der aktuellen Policy, mit der der Agent exploriert. Das bedeutet, auch wenn der Agent zufällige oder suboptimale Aktionen ausprobiert, wird die Q-Update-Regel immer der höchste geschätzte Wert für den nächsten Zustand verwendet. Dadurch nähert sich die gelernte action-value Funktion $Q$ direkt der optimalen action-value Funktion $q^*$ an, unabhängig vom Verhalten des Agenten während des Lernens [11, S. 131].

Sie ist gut geeignet für kleine, diskreten Zustands- und Aktionsräume und ist einfach zu implementieren, aber skaliert schlecht bei großen oder kontinuierlichen Zustandsräumen. Häufig passiert das, wenn zu viele Zustandsvariablen existieren oder die der Zustandsraum in einer zu feinen Auflösung aufgestellt wird.

### 2.3.2 Deep Q-Learning

DQL ist eine Erweiterung des Q-Learnings, welche die Q-Tabelle durch ein tiefes neuronales Netz, dem Policy-Netzwerk, welches $Q_\theta(s, a)$ modelliert, ersetzt. Um dieses Problem zu lösen, approximiert man die Q-Tabelle mit einer parametrischen Funktion $Q_\theta(s, a)$, die für ein gegebenes Zustand-Aktions-Paar den entsprechenden Q-Wert liefert.

Da die Parameter des Policy-Netzes bei jedem Update angepasst werden, ändern sich auch die Zielwerte ständig. Dies führt zu Instabilität im Training, da das Netzwerk versucht sich an seine eigene, sich ständig ändernden Zielwerte anzupassen. Um diese Instabilität zu verringern, führt DQL ein zweites separates Netzwerk, das Target-Netzwerk $Q_{\theta^-}(s, a)$. Sie ist eine Kopie des Policy-Netzes, wird jedoch nur in bestimmten Abständen aktualisiert. Die Zielwerte werden folgend berechnet:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

Somit bleiben die Zielwerte für mehrere Updates konstant und das Training wird stabiler.

Das Ziel des Policy-Netzes ist seine Schätzung $Q_\theta(s, a)$ an den Zielwert $y$ anzupassen. Hierzu wird eine Loss-Funktion definiert, welche angibt wie nah die Schätzung zum Zielwert ist. Die Update-Formel vom DQL sieht entsprechend folgend aus:

$$Q_\theta(s, a) \leftarrow Q_\theta(s, a) + \alpha\big[r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a)\big]$$

**Dabei gilt:**

| | |
|---|---|
| $Q_\theta(s, a)$ | Geschätzter Wert der Aktion $a$ im Zustand $s$ (Policy-Netz) |
| $\max_{a'} Q_{\theta^-}(s', a')$ | Max. geschätzter Wert im Folgezustand $s'$ über alle $a'$ (Target-Netz) |

Die Idee ist analog zum klassischen Q-Learning-Update, aber verwendet anstelle einer Q-Tabelle zwei neuronales Netze.

Eine weitere Komponente, um das Lernen zu stabilisieren, ist der Experience Replay Buffer. Der Experience Replay Buffer speichert vergangene Erfahrungen in Form von Übergängen $(s, a, r, s')$ in einem Speicherpuffer.

Eine Möglichkeit, das Lernen zu stabilisieren, ist der Experience Replay Buffer, der vergangene Übergänge $(s, a, r, s')$ speichert. Zufällige Stichproben aus dem Puffer reduzieren Korrelationen zwischen aufeinanderfolgenden Erfahrungen. Beim Prioritized Experience Replay (Prioritized Experience Replay (PER)) werden Übergängen nach ihrem Lernfortschritt Prioritäten zugewiesen, sodass besonders informative Erfahrungen häufiger genutzt werden.

Eine Episode Training (siehe Algorithmus 2) eines DQL-Agents ist in den zwei Phasen Sampling und Training eingeteilt.

**Algorithm 2** Deep Q-Learning (DQL)

Initialize $Q_\theta$, target $Q_{\theta^-}$, replay buffer $\mathcal{D}$
**for** episode = 1 to $N$ **do**
  Initialize state $s$
  **while** not terminal **do**
    Choose $a$ via $\epsilon$-greedy; execute $a$, observe $r, s'$
    Store $(s, a, r, s')$ in $\mathcal{D}$; $s \leftarrow s'$
    Sample mini-batch from $\mathcal{D}$
    compute target $y \leftarrow r + \gamma \max_{a'} Q_{\theta^-}(s', a')$
    compute loss $L \leftarrow (y - Q_\theta(s, a))^2$
    Update $\theta$ by gradient descent on $L$
    **if** step mod $C = 0$ **then** update $Q_{\theta^-} \leftarrow Q_\theta$
    **end if**
  **end while**
**end for**

In der ersten Phase Sampling werden Aktionen mit $\epsilon$-greedy ausgewählt. Diese werden ausgeführt und der Übergang im Replay Buffer gespeichert.

Bei der zweiten Phase Training wird zufällig eine Mini-Batches vom Replay Buffer entnommen, um das Policy-Netzwerk zu trainieren. Anschließend werden die Parameter des Policy-Netzwerkes und periodisch die vom Target-Netzwerk angepasst.

# 3 Methodik und Experimentaufbau

In diesem Kapitel werden die verwendeten Methoden sowie der Aufbau der Experimente beschrieben, mit denen das Empfehlungssystem zur Musikentdeckung entwickelt wurde.

Zur Beantwortung der Forschungsfrage wurde eine experimentelle Untersuchung mit einer eigens entwickelten Simulationsumgebung durchgeführt. Ziel ist der Vergleich klassischer und RL-basierter Methoden für Musikempfehlungssysteme mit besonderem Hinblick auf ihren Einsatz zur Musikentdeckung.

Zunächst werden die zentralen Annahmen dargestellt, die den gewählten Anwendungsfall des Empfehlens bislang unbekannter Songs bestimmen. Darauf aufbauend wird begründet, warum klassische und RL-basierte Verfahren zur Bearbeitung dieser Aufgabe geeignet sind.

Im Anschluss folgen die Beschreibung der Datengrundlage, der Aufbau der Simulationsumgebung sowie die Implementierung der beiden Empfehlungssysteme. Abschließend werden der Ablauf des Experiments und die verwendeten Evaluationsmetriken dargelegt.

## 3.1 Methodik

Fur die gesamte Arbeit werden folgende Annahmen getroffen, die sowohl die Simulationsumgebung als auch die Datenstruktur betreffen:

**Musikentdeckung im Fokus:** Die Arbeit konzentriert sich auf Empfehlungen neuer, bislang unbekannter Songs. Bereits bekannte Songs werden aus dem Empfehlungsraum ausgeschlossen.

**Content-basierte Empfehlung:** Empfehlungssysteme verwenden ausschließlich Audio-Embeddings. Meta-Merkmale (Künstler, Erscheinungsjahr, Länge, Popularität) werden von den Systemen nicht genutzt.

**Klare Genre-Abgrenzungen:** Jeder Songs wird eindeutig einem Genre zugeordnet. Genre-Überlappungen oder Mischformen existieren zwar im Datensatz.

**Keine situativen Effekte:** Situative Faktoren wie z. B. Tageszeit oder Stimmung werden nicht modelliert.

**Aktives Zuhören:** Nutzer hören aktiv jeden empfohlenen Song und geben ein akkurates Feedback, welches sich in positive oder negativ einteilen lässt.

**Referenzsong-basiertes Verhalten:** Jeder Nutzer hat einen Song, der seine Präferenzen repräsentiert. Es wird angenommen, dass Nutzer Songs bevorzugen, die dem Referenzsong klanglich ähnlich sind und dem gleichen Genre angehören. Nutzer mit mehreren Genre-Präferenzen oder komplexen Geschmacksmustern werden nicht modelliert. Anforderungen gegenüber Metadaten wie Länge oder Erscheinungsjahr wirken minimal auf die Bewertung aus. Positives Feedback erfordert also eine ausreichende Ähnlichkeit und grobes Einhalten der Metadatenbedingungen.

### 3.1.1 Auswahl der Algorithmen

Für diese Arbeit wurde ein experimenteller Ansatz gewählt, der den Vergleich zweier Empfehlungssysteme ermöglicht: ein klassisches, auf k-NN basierendes Modell und ein Reinforcement-Learning-Ansatz mit DQL. Beide Systeme werden auf einer selbst entwickelten Simulationsumgebung mit denselben synthetischen Nutzern getestet. Somit können beide Methoden unter identischen Bedingungen evaluiert und Unterschiede in der Leistung auf die jeweiligen Algorithmen zurückgeführt werden.

Die Wahl von k-NN als klassisches Verfahren begründet sich durch seine Eignung für content-basierte Empfehlungen [10, S. 2], besonderes der Vertiefung der Nutzerpräferenzen. Über die Ähnlichkeit der Song-Embeddings lassen sich Songs identifizieren, die dem Geschmack des Nutzers entsprechen, ohne dass ein komplexes Trainingsverfahren notwendig ist. Ihre Anpassungsfähigkeit ist begrenzt und ist auf einer akkuraten Modellierung ihrer Nutzerinteressen abhängig.

Als repräsentativer RL-Ansatz wurde Deep Q-Learning (DQL) gewählt, um das Potenzial lernbasierter Methoden in einem kontinuierlichen und großen Zustandsraum zu untersuchen. DQL ermöglicht den Einsatz von $\epsilon$-greedy Strategien, wodurch ein exploratives Verhalten des Agenten gefördert wird. Die Methode erlaubt es, die Präferenzen eines Nutzers zu erfassen, auch wenn diese nicht direkt aus dem bisherigen Hörverlauf abgeleitet werden

können. Die Wahl fiel auf DQL aufgrund seiner Fähigkeit, mit kontinuierlichen Zuständen umzugehen, seiner explorativen Natur und der vergleichsweise schnellen Trainingszeit.

Damit bieten k-NN und DQL eine ausgewogene Balance zwischen Praktikabilität, Vergleichbarkeit und Aussagekraft für die Untersuchung von Musikempfehlungen.

### 3.1.2 Datengrundlage

Der Datensatz umfasst 1200 Songs, gleichmäßig verteilt auf die Genres Klassik, Jazz, EDM, Hip-Hop, Pop und Rock. Die Informationen zu Titel, Künstler, Erscheinungsjahr, Dauer und Popularität wurden über die Spotify-API bezogen, während die 30-Sekunden-Audioausschnitte von der Deezer-API bereitgestellt wurden. Die Songs wurden aus sechs von Spotify kuratierten Playlists ausgewählt, die jeweils ein Genre repräsentieren.

Tabelle 3.1 bietet einen Überblick über zentrale Metadaten der Songs im Datensatz. Sie zeigt, dass die Songs verschiedene Veröffentlichungsjahre, Längen, Popularitätsgrade und Künstler abdecken. Im Anhang sind die Metadaten zusätzlich nach Genre aufgeschlüsselt.

| Merkmal | min | max | avg. | std | Beschreibung |
|---|---|---|---|---|---|
| Jahr | 1954 | 2025 | 2011.74 | 16.81 | Erscheinungsjahr |
| Song-Länge | 1:21 | 27:17 | 3:57 | 1:43 | Minuten:Sekunden |
| Popularität | 17 | 96 | 62.39 | 16.1 | Ranking (0–100) |
| Künstler | 1 | 11 | 1.46 | 1 | Häufigkeit pro Künstler |

Tabelle 3.1: Übersicht der Metadaten der Songs im Datensatz (Minimum, Maximum, Mittelwert, Standardabweichung)

Für die Repräsentation der Songs wurde zunächst Audio-Embeddings mit OpenL3 erzeugt. Die OpenL3-Publikation [12] zeigt, dass die OpenL3-Embeddings semantische Eigenschaften von Audiosignalen erfassen können. Das vortrainierte OpenL3-Modell liefert zuverlässige Ergebnisse, selbst wenn die Art der Daten von den Trainingsdaten abweicht. Dies rechtfertigt ihren Einsatz für inhaltsbasierte Musikempfehlungen auf synthetischen Nutzerdaten.

Die Audio-Embeddings basieren nur auf den ersten 30 Sekunden eines Songs und werden im Rahmen dieser Arbeit zusätzlich von 512 auf 32 Dimensionen komprimiert. Durch diese Kürze werden der Songanfang und mögliche Intro-Passagen erfasst, während der restliche Verlauf unberücksichtigt bleibt. Insbesondere bei Songs mit zunächst ähnlicher

Instrumentierung oder Harmonik kann dies dazu führen, dass Lieder, die sich später stark unterscheiden, in den Embeddings als ähnlich bewertet werden.

Ein Beispiel hierfür ist ein klassisches Klavierstück, dessen Anfang einem Jazz-Stück ähnelt. Während die Embeddings für die ersten 30 Sekunden eine hohe Ähnlichkeit zeigen, ändert sich die Musik im weiteren Verlauf deutlich. Dieses Phänomen wird durch die Nutzung von Deezer-Vorschauen verstärkt, da vornehmlich bei unbekannten Songs häufig nur die ersten 30 Sekunden verwendet werden. Daher spiegeln die Embeddings primär lokale Ähnlichkeiten wider und nicht die gesamte musikalische Struktur eines Songs.

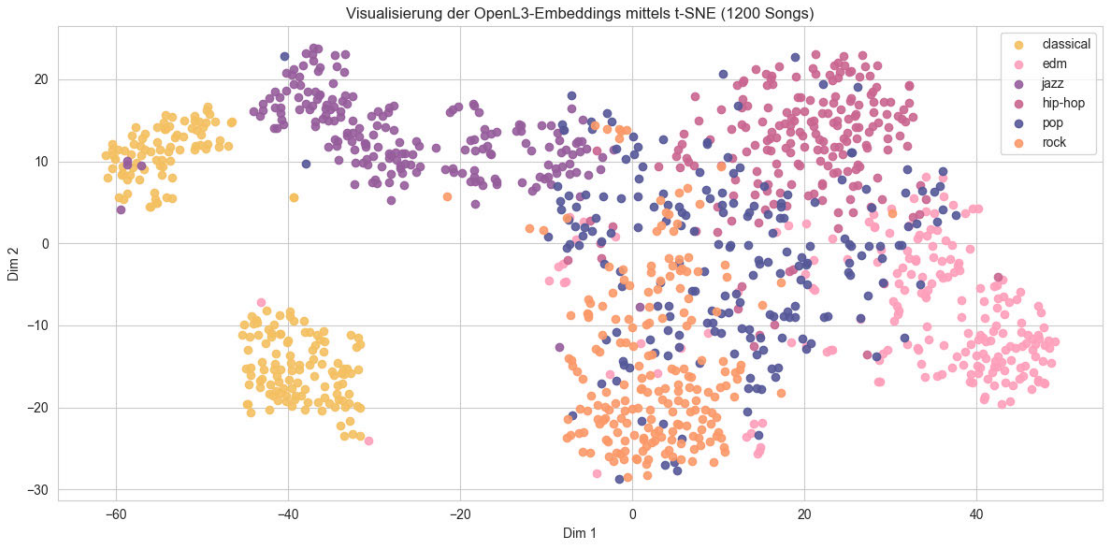



Abbildung 3.1: t-SNE Visualisierung der OpenL3-Embeddings. Jeder Punkt entspricht einem Song, Farben kennzeichnen Genres. Klanglich ähnliche Songs sind näher gruppiert.

In Abbildung 3.1 ist zu erkennen, dass Songs der Genres Klassik, Jazz, Hip-Hop, EDM und Rock kompakte Cluster bilden. Lediglich EDM und Hip-Hop überlappen sich teilweise mit Pop, was auf akustische Ähnlichkeiten zurückzuführen ist.

Zum Vergleich wurden klassische Audio-Features mit Librosa extrahiert. Librosa[13] ist eine Python-Bibliothek für Musik- und Audioanalyse, die auf Signalverarbeitung basiert. Typische Features wie MFCCs, eignen sich besonders für die Genre-Klassifikation. Diese Merkmale wurden zunächst aus 1200 Songs berechnet und zu einem Feature-Vektor kombiniert.

Im Anhang verdeutlicht Abbildung A.1, warum OpenL3 als Hauptrepräsentation für die Empfehlungssysteme genutzt wurde. Während die Librosa-Features nur Klassik und Jazz klar von den anderen Genres trennen, überschneiden Rock, Pop, Hip-Hop und EDM stark. Die OpenL3-Embeddings zeigen eine deutlich bessere Trennung der Genres im Merkmalsraum, besonders für EDM, Hip-Hop und Rock.

Die Tabelle 3.2 zeigt, dass klassische Musik und Hip-Hop innerhalb des Genres sehr ähnlich klingen, während bei Pop und Jazz klanglichen Unterschiede vorweisen. Extremwerte zeigen, dass alle Genres sowohl sehr ähnliche als auch stark unterschiedliche Songs enthalten.

| Genre | min | max | avg. |
|---|---|---|---|
| classical | -0.25 | 0.99 | 0.56 |
| edm | -0.76 | 0.96 | 0.38 |
| hip-hop | -0.44 | 0.96 | 0.59 |
| jazz | -0.66 | 0.97 | 0.30 |
| pop | -0.83 | 0.95 | 0.25 |
| rock | -0.74 | 0.98 | 0.34 |

Tabelle 3.2: Durchschnittliche, minimale und maximale Cosine-Similarity der Songs innerhalb jedes Genres.

Die Tabelle 3.3 zeigt die durchschnittliche Anzahl an Songs pro Genre innerhalb von drei Ähnlichkeitsbereichen basierend auf den Song-Embeddings. Die Bereiche zeigen stark unähnliche, moderat ähnliche und stark ähnliche Songs innerhalb des jeweiligen Genres. Man kann hier besonders die unterschiedliche Verteilung der Songs erkennen, die mit der Natur des Genres gerecht ist. Zum Beispiel sind haben Pop Songs im Durchschnitt nur zwölf stark ähnliche Songs, aber 142 moderate ähnliche Songs.

| Genre | unähnlich | moderat ähnlich | stark ähnlich |
|---|---|---|---|
| classical | 3.03 | 121.76 | 74.21 |
| edm | 28.69 | 136.64 | 33.67 |
| hip-hop | 3.82 | 125.10 | 70.08 |
| jazz | 47.06 | 122.11 | 29.83 |
| pop | 43.81 | 142.86 | 12.33 |
| rock | 26.32 | 136.10 | 36.58 |

Tabelle 3.3: Durchschnittliche Anzahl unähnlicher, moderat ähnlicher und stark ähnlicher Songs pro Song im Genre

Die Figure A.2 visualisiert die Ähnlichkeiten der Tabelle. Besonders zu erkennen ist, dass klassische Musik eine starke Homogenität vorweist, aber trotzdem leicht negative

Ähnlichkeiten zu bestimmte Songs hat, weil diese sich an zwei Punkten clustern. Das liegt daran, dass die verwendeten Audiospuren von einer ganzen Orchestra Symphonie oder primär von Pianostücken dominiert werden. Auch von Pianostücken geprägt sind auch viele Jazz-Songs, weshalb beide Genres sehr weit von EDM oder Hip-Hop dargestellt sind, die hauptsächlich elektrische Instrumente verwenden.

Um die abstrakten Zahlen greifbar zu machen, zeigt Tabelle 3.4 die Songpaare mit extrem hoher und niedriger Ähnlichkeit.

| Genre | Ähnlichsten Songs | Unähnlichsten Songs |
|---|---|---|
| classical | Scriabin: 24 Preludes, Op. 11: No. 21 in B-Flat Major & Frost - Grand Piano | William Tell Overture & Piano Concerto in G Major, M. 83: II. Adagio assai |
| edm | No Division (feat. XSALT) & Reverie | Children & Head & Heart (feat. MNEK) |
| hip-hop | MIDDLE CHILD & Time Today | Just Wanna Rock & Plain Jane |
| jazz | Moonlight in Vermont & Why Did I Choose You | The Rose Tattoo & Love for Sale |
| pop | TiK ToK & Give Me Everything (feat. Nayer) | Black Magic & Hometown Glory |
| rock | Enter Sandman - Remastered 2021 & Master Of Puppets | Rollin' (Air Raid Vehicle) & Bohemian Rhapsody - Remastered 2011 |

Tabelle 3.4: Beispiele für extrem ähnliche und unähnliche Songs innerhalb eines Genres

### 3.1.3 Simulationsumgebung

Um die Empfehlungsansätze vergleichbar zu evaluieren, wurde eine Simulationsumgebung entwickelt, die Nutzerinteraktionen abbildet und Feedback für Empfehlungen gibt. In der Simulation werden einzelne Empfehlungen gegeben und Rückmeldungen erhalten, weil Musik in der Realität häufig in Form von aufeinanderfolgenden Songs konsumiert wird (z. B. Playlists, Radios, Alben).

Sie bietet Schnittstellen an, wodurch Empfehlungssysteme ein Nutzerprofil erhalten und darauf basierend ihre Songempfehlung generieren. Ein Nutzerprofil wird als mehrdimensionalen Vektor dargestellt, der den gewichteten Mittelwert der Song-Embeddings aller stark positiv bewerteten Songs bildet. Dazu erhalten die Systeme die Song-Embedding des zuletzt empfohlenen Song und die letzten zwei Feedbacks als One-Hot-Encoding.

Über der Simulationsumgebung werden dem Nutzer die Songempfehlungen simulativ durchgespielt und dem Empfehlungssystem eine Rückmeldung gegeben.

$$R(a) = \begin{cases} 2.0 + b, & \text{wenn feedback}(a) = \text{save} \\ 1.0 + b, & \text{wenn feedback}(a) = \text{listen-until-end} \\ -0.5, & \text{wenn feedback}(a) = \text{skip} \\ -1.0, & \text{wenn feedback}(a) = \text{dislike} \end{cases} \tag{3.1}$$

wobei

$$b = \begin{cases} 0.5, & \text{wenn feedback}(a) \text{ positiv ist (save oder listen-until-end)} \\ 0, & \text{sonst} \end{cases}$$

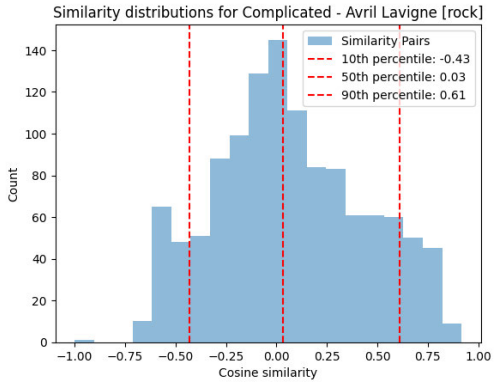
**Nutzersimulation**

Zur Bewertung der Empfehlungen wird eine realistische Nutzersimulation als Grundlage der Offline-Evaluation eingesetzt. Für die Simulation der Nutzer wurden synthetische Profile erzeugt, die individuellen Präferenzen abbilden. Dabei werden neuere Songs leicht stärker gewichtet, um aktuelle Interessen stärker zu berücksichtigen.

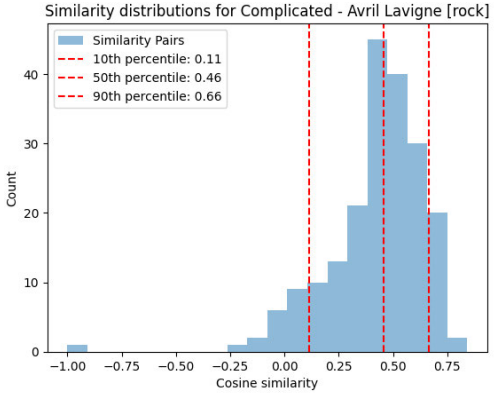
Die Bewertung eines Songs basiert auf einer Kombination aus inhaltsbasiertem und regelbasiertem Feedbackmechanismus. Dabei bleibt die Funktionsweise für das Empfehlungssystem verborgen. Jeder Nutzer erhält einen Referenzsong, der seine Präferenzen repräsentiert. Die Ähnlichkeit eines Songs zu diesem Referenzsong bestimmt maßgeblich die inhaltsbasierte Bewertung:

- Songs, die dem Referenzsong klanglich stark ähneln, werden positiv bewertet,
- während unähnliche Songs negativ eingeordnet werden.

Der inhaltsbasierte Teil bewertet Songs anhand der klanglichen Ähnlichkeit zu einem Referenzsong mittels Cosine-Similarity. Durch das Festlegen eines Songs, welche als Intention des Nutzers interpretiert werden kann, können deterministische Feedbacks generiert werden. Weil die Abstände jeder Songs mit den Song-Embeddings statisch sind, kann zu jedem Song die Ähnlichkeit zum Referenzsong zuverlässig berechnet werden.




(a) Ähnlichkeitsverteilung von allen Songs im Datensatz zu diesem.




(b) Ähnlichkeitsverteilung von allen Songs im gleichen Genre zu diesem.

Abbildung 3.2: Das Histogramm zeigt die Verteilung der Ähnlichkeitswerte aller Songs zu einem ausgewählten Referenzsong. Auf der x-Achse ist die Ähnlichkeit, auf der y-Achse die Anzahl der Songs dargestellt.

Mittels der Histogramme (siehe Abbildung 3.2) kann man einschätzen, wie viele Songs einem Song ähnlich sind und entsprechend einen Schwellenwert definieren. Dieser interpretiert für den Nutzer den Punkt, an dem Songs „ähnlich genug“ klingen, um konsumiert zu werden.

Die Ähnlichkeit $s_{\text{sim}}$ eines Songs zum Referenzsong wird über die Cosine-Similarity berechnet und anschließend linear auf einen Score abgebildet (Gleichung 3.2). Hierbei gilt $\tau_{\text{high}} = 1$, sodass alle Songs, die maximal ähnlich zum Referenzsong sind, den höchsten Score erhalten. $\tau_{\text{low}}$ definiert die untere Grenze, ab der Songs als überhaupt „ähnlich genug“ für positives Feedback gelten. Die Ähnlichkeit wird linear auf einen Score $s_{\text{sim}}$ abgebildet:

$$s_{\text{sim}} = \begin{cases} -1 + \dfrac{\text{sim}}{\tau_{\text{low}}}, & \text{wenn sim} \leq \tau_{\text{low}} \\ \dfrac{\text{sim} - \tau_{\text{low}}}{\tau_{\text{high}} - \tau_{\text{low}}}, & \text{wenn } \tau_{\text{low}} < \text{sim} \leq \tau_{\text{high}} \end{cases} \tag{3.2}$$

Der regelbasierte Teil bewertet Songs anhand von Genre-Präferenzen, Erscheinungsjahr und Länge. Songs außerhalb der bevorzugten Genres werden stark negativ bewertet, während innerhalb der präferierten Genres die Bewertung durch Toleranzen für Länge und Jahr angepasst wird:

$$s_{\text{meta}} = \begin{cases} 2, & \text{wenn Genre, Jahr und Länge passend} \\ 1 \text{ oder } 0, & \text{wenn Genre und Länge passend (50/50 Zufall)} \\ 1 \text{ oder } -1, & \text{wenn Genre und Jahr passend (30/70 Zufall)} \\ 0 \text{ oder } -1, & \text{wenn nur Genre passend (gleichverteilt)} \\ -2, & \text{sonst} \end{cases} \tag{3.3}$$

Die beiden Komponenten werden zu einem Gesamt-Score $f$ kombiniert:

$$f = (1 - w_{\text{sim}}) \cdot s_{\text{meta}} + 2 \cdot w_{\text{sim}} \cdot s_{\text{sim}} \tag{3.4}$$

mit

- $s_{\text{meta}}$ = regelbasiertes Feedback (3.3)
- $s_{\text{sim}}$ = klangbasiertes Feedback (3.2)
- $w_{\text{sim}}$ = Gewichtungsfaktor, wobei $w_{\text{sim}} \in [0.5, 0.7]$.

Der Gewichtungsfaktor $w_{\text{sim}}$ bestimmt den Anteil der klangbasierten Komponente am Gesamtfeedback und wird typischerweise zufällig aus dem Intervall [0.5, 0.7] gezogen. So wird die Ähnlichkeit der Audio-Features stärker gewichtet als die Metadaten. Die konkrete Genreverteilung spielen dabei weiterhin eine wichtige Rolle, sodass sehr ähnliche Songs unterschiedlicher Genre nicht zu stark positiv bewertet werden können.

Basierend auf dem kombinierten Score $f$ wird der Song einer von vier möglichen Reaktionen zugeordnet:

$$FB(f) = \begin{cases} \text{stark positiv}, & f > 1 \\ \text{positiv}, & 0 \leq f \leq 1 \\ \text{negativ}, & -1 \leq f < 0 \\ \text{stark negativ}, & f < -1 \end{cases} \tag{3.5}$$

Diese vier Reaktionen in Tabelle 3.5 lassen sich außerdem in zwei positive und zwei negative Kategorien unterteilen.

| Feedbacktyp | Bedeutung | Bewertung |
|---|---|---|
| save | speichert Song, um erneut zu hören | stark positiv |
| listen-until-end | hat Song gehört, nicht interessiert erneut zu hören | positiv |
| skip | überspringt Song wegen fehlenden Interesses | negativ |
| dislike | möchte diese Musik aktiv vermeiden | stark negativ |

Tabelle 3.5: Mögliche Nutzerreaktionen auf Songs. Die Zuordnung erfolgt über den kombinierten Score $f$ (Gleichung 3.4).

Der kombinierte Score $f$ bewertet Songs anhand von zwei Komponenten: klangliche Ähnlichkeit und Metadaten. Ein Song wird als positiv (listen-until-end) gewertet, wenn mindestens eine der beiden Komponenten ausreichend hoch ist. Für stark positives Feedback (save) müssen beide Komponenten hoch sein: Der Song muss genug klanglich ähnlich zum Referenzsong sein und den Genre- sowie Metadatenpräferenzen des Nutzers entsprechen. Songs, die nur in einer Komponente gut abschneiden, erhalten im schlechtesten Fall nur positives Feedback (listen-until-end). Songs, die in beiden Komponenten schlecht abschneiden, werden negativ bewertet.

### Synthetische Erstellung der Nutzerdaten

Da das Ziel der Arbeit darin besteht, zu vergleichen, wie die Empfehlungssysteme mit unterschiedlichen Nutzerverhalten umgehen, werden zwei Arten von Nutzern definiert:

- Präferenztreue Nutzer (korrekte Profil): Historien stimmen vollständig mit den individuellen Interessen der Nutzer (Referenzsong) überein.
- Explorative Nutzer (inkorrektes Profil): Historien stimmen nur mit dem Genre der Interessen der Nutzer (Referenzsong) überein.

Um diese Nutzer zu simulieren, müssen ihre Hörverläufe in Berücksichtigung zum Referenzsong sein. Diese werden pro Genre synthetisch erzeugt nach Algorithmus 3 erzeugt. Für jedes Genre wird die Menge der validen Referenzpunkte berechnet. Ein Referenzpunkt gilt als valide, wenn genügend Songs vorhanden sind, um einen passenden Hörverlauf zu generieren. Zudem müssen noch ausreichend positive Songs für die Simulation übrig bleiben.

**Algorithm 3** User History Generation

**for** each user $u$ **do**
    Randomly select a reference song $s_{\text{ref}}$ from the genre's song list
    Compute pairwise similarity of this embeddings to all songs in the genre
    **if** user type $u$ = "correct" **then**
        Select songs as the most similar to $s_{\text{ref}}$
    **else if** user type $u$ = "incorrect" **then**
        Select songs slightly deviating from $s_{\text{ref}}$
    **end if**
    Adjust the history length to a fixed number of songs
    Adjust lowest similarity threshold to tighten the number of candidate songs
    Store the history for user $u$
    Count remaining songs for proof
**end for**

Diese Konstruktion ermöglicht es, Profile zu erzeugen, deren Historien eng am Referenzsong ausgerichtet sind und somit präferenz-getreues Verhalten widerspiegeln. Profile, die aus den unähnlichsten Songs zum Referenzsong bestehen, bilden dagegen explorierendes Verhalten ab. Da jedes Genre nur 200 Songs besitzt, wurde darauf geachtet, keine Referenzsongs zu wählen, die sich nicht am Rande eines Genres befinden. Somit wird auch sichergestellt, dass alle Nutzer immer eine ausreichende Anzahl an Songs besitzen, die sie positiv bewerten können.

### 3.1.4 Modellierung der Empfehlungssysteme

Der folgende Abschnitt beschreibt die beiden in dieser Arbeit entwickelten Empfehlungssysteme: ein klassisches, auf Ähnlichkeitsberechnung basierendes Verfahren sowie ein modellbasiertes Verfahren auf Basis von Reinforcement Learning. Beide Systeme greifen auf dieselbe Datengrundlage und Umgebungslogik zurück. Der Unterschied liegt in der Art und Weise, wie Empfehlungen erzeugt werden.

#### Klassisches Modell (k-NN)

Das klassische Empfehlungssystem verwendet k-NN, um über Ähnlichkeitssuche eine Regressionsvorhersage zur Relevanz eines Songs für einen Nutzer zu treffen. Sie bezieht

sich auf die Annahme, dass der Nutzer seine aktuellen Präferenzen vertiefen möchte und wird gezielt Empfehlungen geben, die sehr ähnlich zum Nutzerprofil sind. Da das Modell keine Lernkomponente besitzt, bleibt das Modell statisch und nutzt nur das Nutzerprofil. Nach jeder Empfehlung wird der Songkatalog aktualisiert und k-NN auf die verfügbaren Songs angepasst.

Zur Generierung von Empfehlungen wird die Ähnlichkeit zwischen dem Zustandsvektor und dem Embeddings aller verfügbaren Songs bzw. unbekannte Songs berechnet. Als Distanzmaß wird die Cosine-Distanz verwendet, die eine lineare Transformation der Cosine Similarity darstellt, um sie als eine Distanzmetrik zu verwenden. Anschließend wird dem Nutzer den Song mit der geringsten Distanz empfohlen.

#### RL-basiertes Empfehlungssystem

Für das RL-Empfehlungssystem wird ein DQN trainiert. Ziel ist es, eine Policy zu erlernen, die zu jedem Zeitpunkt den bestmöglichen Song empfiehlt, sodass die erwartete Belohnung des simulierten Nutzers maximiert wird. Konkret muss sie eine Policy lernen, die Songs so auswählt, dass der Nutzer über die Session interessiert und zufrieden bleibt.

Die Empfehlungsaufgabe lässt sich folgend als RL-Problem definieren. Der Agent ist das Empfehlungssystem und die Umgebung ist die Simulationsumgebung mit der Nutzersimulation, die Feedback zu Empfehlungen gibt. Ein Zustand der Simulationsumgebung ist eine Kombination aus dem Geschmacksprofil des Nutzers, der letzte empfohlene Song-Embedding, der zuletzt erhaltene Feedback und dem Interesse-Wert des Nutzers. Eine Aktion ist die Auswahl eines Songs aus dem Katalog als Empfehlung und die Bewertung der Empfehlung bestimmt die Belohnung bzw. Bestrafung.

Das zentrale Prinzip besteht darin, dass das neuronale Netz die Q-Funktionen approximiert. Diese ordnet jedem möglichen Zustand-Aktions-Paar einen Wert zu, der die erwartete zukünftige Belohnung beschreibt. Durch das ständige Aktualisieren der Q-Werte anhand des Nutzerfeedbacks lernt das Modell, welche Empfehlungen in welchen Zuständen vorteilhaft sind. Im Gegensatz zum tabellen-basiertem Q-Learning, lernt DQN ähnliche Zustand-Aktions-Paare mit ähnlichen Q-Werten zu approximieren und kann dadurch auch für bisher unbekannten Zustände sinnvolle Entscheidungen ableiten. Dadurch lernt DQN übergeordnete Präferenzen des Nutzers sowie die Songs, die mit diesen Präferenzen korrelieren. Da DQN einen diskreten Aktionsraum erwartet, lernt das Netz nur welche

Song-IDs zu welchem Zustand passen, um Belohnungen zu erhalten. Indirekt lernt es das simulierte Verhalten.

Speziell für den RL-basierten Ansatz enthält die Simulationsumgebung eine Reward-Funktion (siehe Gleichung 3.1), die das Nutzerfeedback auf numerische Belohnungen oder Bestrafungen abbildet. Positives Feedback führt zu einer Belohnung, während negatives Feedback eine Strafe nach sich zieht. Da die Empfehlung von neuen Songs in erster Linie das Vermeiden negativer Rückmeldung priorisiert, gibt es zusätzlich ein Bonus, wenn mehrere positive Rückmeldungen aufeinander folgen. Somit wird der RL-Agent motiviert, fortlaufend erfolgreiche Empfehlungsketten zu erzeugen.

Das DQN Modell ist ein einfaches Feedforward-Neuronales Netzwerk, implementiert in PyTorch [14]. Es besteht aus drei fully-connected Layers mit ReLU-Aktivierung zwischen ihnen und einem PER-Buffer. Die Netzwerkstruktur wurde bewusst einfach gehalten, um den Fokus auf das Interaktionsverhalten und die Umgebungsmodellierung zu legen. Eingabe ist der Zustandsvektor des Nutzers, dass aus seinem Geschmacksprofil (32 Dims), die Song-Embedding der letzten Historie (32 Dims) und das One-Hot Encoding der letzten zwei Feedbacks (jeweils 4 Dims). Ausgabe sind die Q-Werte für alle Songs, wobei nur der unbekannte Song mit dem höchsten Q-Wert verwendet wird.

Deep Q-Network Training

Für das Experiment wurde ein DQN-Modell trainiert, dass auf eine gemischte Nutzerschaft von akkurate abgebildete und leicht verschobenen Profilen abgestimmt ist. Eine Episode besteht aus 20 Empfehlungen vom Agenten an den simulierten Nutzer. Ziel des Agenten ist es, sequentiell Songs zu empfehlen, die positiven Feedback generieren.

Der Trainingsdatensatz besteht aus 1479 unterschiedliche Nutzern. Das Modell wurde über 3000 Episoden mit jeweils 500 Nutzern pro Genre trainiert. Es wird ausgeschlossen, dass Nutzer am Anfang keine positiven Bewertungen abgeben. Mehrere Testläufe haben gezeigt, dass das Modell bei 3000 Episoden kontinuierlich lernt und ab 3500 Episoden der Speicher des Replay Buffers überlastet wird.

Der Epsilon-Decay wurde so gewählt, dass Epsilon gegen Ende von 75% der Trainingsepisoden sein Minimum erreicht. Dadurch wird zum zunächst Exploration und später Exploitation gefördert. Im Fokus steht nicht die Optimierung des DQN, sondern der Vergleich seiner Anpassungsfähigkeit an unterschiedliche Nutzerpräferenzen. Daher werden diese DQN-Hyperparameter konstant gehalten.

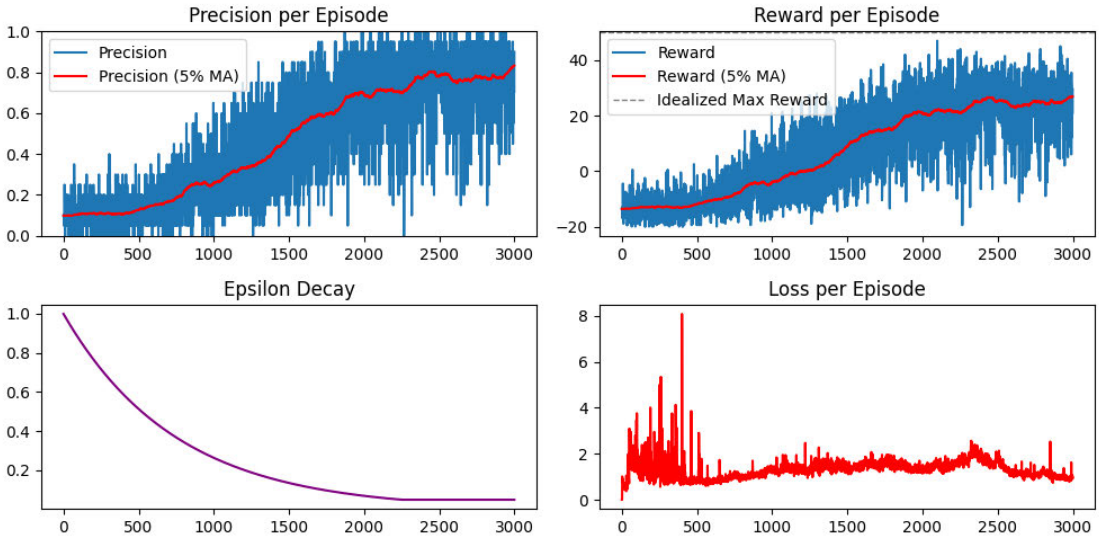



Abbildung 3.3: Trainingsverlauf des DQN-Agenten über 3000 Episoden mit 20 Empfehlungen pro Episode. Oben links: Präzision zeigt, wie gut der Agent die Nutzerpräferenzen trifft. Oben rechts: durchschnittlicher Reward als Maß für die kumulierte Belohnung. Unten links: Epsilon-Verlauf der $\epsilon$-greedy-Strategie. Unten rechts: Loss-Kurve des Netzwerks.

Abbildung 3.3 zeigt den Trainingsverlauf des DQN-Modells. Präzision und Reward steigen im Verlauf an, während Loss und Epsilon sinken, was den Übergang vom explorativem zum exploitativem Verhalten zeigt.

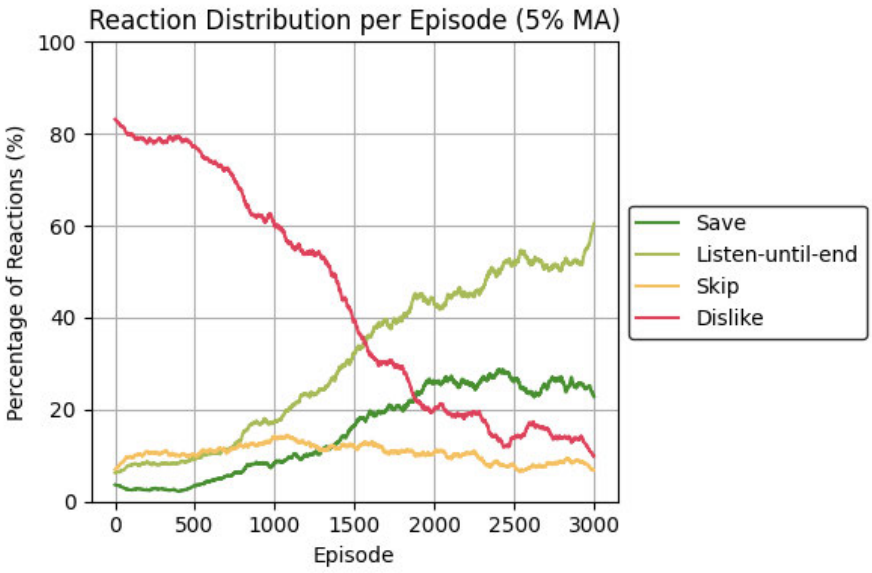



Abbildung 3.4: Gleitender 5%-Mittelwert der Anteile der Nutzerreaktionen pro Episode. Die Linien zeigen den Anteil der vier Feedbacktypen (save, listen-until-end, skip, dislike) pro Episode.

Abbildung 3.4 stellt die Verteilung der Nutzerreaktionen im Zeitverlauf dar. Während zu Beginn überwiegend negative Reaktionen dominierten, verschieben sich die Reaktionen im Verlauf zu positiven Reaktionen. Die Policy zeigt, dass das Modell versucht die Bestrafung auszugehen. Ihre Policy geht stark positiven Reaktionen hervor und bevorzugt eher positive Reaktionen.

## 3.2 Experimentaufbau

Ziel des Experiments ist es, die Leistungsfähigkeit von klassischen ML-Methoden (hier k-NN) und eines RL-Ansatzes (DQL) für die Musikentdeckung zu vergleichen. Im Mittelpunkt steht die Frage, unter welchen Umständen welche Methode die Präferenzen der Nutzer erfassen kann und Songs erfolgreich empfehlen kann.

Das Experiment prüft zum einen die Robustheit der jeweiligen Methode bei vollständigen und unvollständigen Nutzerprofilen in einer vereinfachten Modellierung. Zum anderen wird geprüft, ob die Exploration und Exploitation von DQL einen Vorteil erbringt, indem

es neue Bereiche des Embedding-Raums der Songs erkundet, während k-NN nur auf Ähnlichkeit setzt.

In der Simulationsumgebung werden zwei Arten von Nutzern modelliert, die jeweils reproduzierbares Feedback zu jedem Song abgeben. Aus Sicht des Empfehlungssystems werden die Gruppen als **„korrekt“** und **„inkorrekt“** bezeichnet: Korrekte Nutzer liefern Feedback, das den vom System erwarteten Präferenzen entspricht, während inkorrekte Nutzer bewusst von diesen Erwartungen abweichen, um neue Musikrichtungen innerhalb ihres Genres zu erkunden. Durch diese Differenzierung lassen sich die Stärken und Schwächen der jeweiligen Empfehlungssysteme systematisch untersuchen.

Als Baseline wird ein genre-gebundene Zufallsauswahl eingesetzt, dass das Nutzerprofil zu einem Genre einordnet und dann von der Menge der Songs des Genres zufällig Songs empfehlt. Dieses Verfahren berücksichtigt also lediglich die Genre-Zugehörigkeit des Profils, jedoch keine individuellen Präferenzen innerhalb des Genres.

### 3.2.1 Durchführung der Experimente

Jede Nutzergruppe wird mit 30 Testnutzern getestet. Die Testnutzer sind gleichmäßig auf die Genres verteilt, jeweils 5 Nutzer pro Genre. Über 20 Empfehlungsschleifen hinweg gibt das System Empfehlungen aus und die Nutzer liefern dafür Feedback.

Es werden explizit Nutzer mit unterschiedlichen Hörverläufen verwendet, die mindestens 20 relevante Songs noch offen haben.

Die Tabelle 3.6 zeigt die durchschnittliche Anzahl positiver Songs pro Nutzer und Genre. Sie verdeutlicht die Herausforderung für das Empfehlungssystem, die aus der unterschiedlichen Klangähnlichkeitsverteilung innerhalb der Genres resultiert. Bei Genres mit wenigen positiven Songs und breiteren Clustern im Embedding-Raum (z.,B. Pop) muss das System aus einer kleinen Auswahl relevanter Songs 20 Empfehlungen generieren. Das erhöht die Wahrscheinlichkeit, dass auch weniger relevante Songs empfohlen werden, und erschwert eine konsistent hohe Präzision. Enge Cluster innerhalb eines Genres (z. B. Rock, Jazz, Hip-Hop) liefern hingegen viele Songs, die als positiv gelten, und ermöglichen stabilere und präzisere Empfehlungen.

Außerdem wurde sichergestellt, dass die Nutzer dem DQN Modell unbekannt sind. Hörverlauf und Reaktionsmuster der Nutzer wurden anhand fester Seeds für Reproduzierbarkeit

| Genre | Avg #Pos. Songs (correct) | Avg #Pos. Songs (incorrect) |
|---|---|---|
| rock | 48.0 | 55.0 |
| pop | 25.0 | 31.6 |
| jazz | 54.4 | 51.0 |
| hip-hop | 55.2 | 45.2 |
| edm | 44.0 | 57.2 |
| classical | 60.2 | 63.0 |

Tabelle 3.6: Durchschnittliche Anzahl an möglichen Empfehlungen für korrekte und inkorrekte Testnutzer pro Genre.

erstellt. Für jede Konfiguration und jedes Modell wurden die Testnutzer kopiert, sodass alle Modelle dieselben Nutzer sahen und die Ergebnisse vergleich blieben.

### 3.2.2 Auswahl der Evaluationsmetriken

Zur Bewertung der Qualität der Empfehlungen wurden verschiedene Metriken von der Übersicht [9] entnommen, die die Leistung der Methoden für das Experiment am besten messen.

Precision ist einer der etabliertesten Metriken in der Evaluierung von Empfehlungssystemen [6] und misst den Anteil relevanter Empfehlungen. Sie gibt an, wie häufig das System mit seinen Vorschlägen tatsächlich die Preference des Nutzers trifft. Formal ergibt sich die Präzision eines Nutzers $u$ aus dem Verhältnis von positivem Feedback zur Gesamtzahl empfohlener Songs:

$$\text{Precision}_u = \frac{\text{Anzahl positives Feedback}_u}{\text{Anzahl Empfehlungen}_u} \tag{3.6}$$

Um das Verhalten des Systems in Detail zu erfassen, wurde ergänzend die Feedback-Verteilung pro Nutzer miteinbezogen. Diese zeigen der konkrete Reaktionstyp eines Nutzers, wodurch gezielt direkte Vergleiche gezogen werden können. Die Formel 3.7 berechnet den Anteil der Feedbacktypen pro Nutzer:

$$p_{fb} = \frac{\text{Anzahl der Vorkommen von} fb}{\text{Gesamtzahl der Feedbacks}} \tag{3.7}$$

Ein weiterer Aspekt ist die Stabilität der Empfehlungen, gemessen über die längste positive Feedback-Streak. Diese erfasst, wie lange ein System in der Lage ist, zusammen-

hängend passende Vorschläge zu generieren. Besonders im Kontext von Musikentdeckung ist eine kontinuierliche Serie relevanter Empfehlung bedeutsam, da sie ein positives Nutzererlebnis signalisiert. Für jeden Nutzer $u$ wird die Länge der längsten Sequenz $s$ ermittelt, um dann den Durchschnitt über alle Nutzer $U$ pro Genre zu ermitteln.

$$\text{Streak} = \frac{1}{N} \sum_{u=1}^{N} s_u \tag{3.8}$$

Schließlich wird die Diversität[15] der Empfehlungen untersucht. Die Metrik beschreibt, wie unterschiedlich die Songs innerhalb einer Empfehlungsliste sind. Sie ist insbesondere wichtig, um die klangliche Vielfalt der Empfehlungen zu messen. Formal wird Diversität über den mittleren Abstand der Song-Embeddings bestimmt, hier gemessen mittels Cosine-Similarity (siehe Gleichung 3.2.2.

$$\text{Diversity}_u = 1 - \frac{2}{n(n-1)} \sum_{i`j} \text{cosine-similarity}(s_i, s_j)$$

(3.9)

# 4 Ergebnisse

Die Reihenfolge der Balken in Abbildung A.6, 4.1 und 4.2 entspricht der Reihenfolge der Nutzer in den Tabellen A.5. Aus Gründen der Konsistenz werden die Nutzergruppen im weiteren Verlauf der Arbeit als „korrekt“ (präferenz vertiefend) und „inkorrekt“ (explorativ) bezeichnet.

## 4.1 Ergebnisse der Experimente per Modell

### 4.1.1 Genre-gebundenen Zufallsauswahl (Random)

| Genre | Präzision | Diversität | Streak |
|---|---|---|---|
| Rock | 0.67 | 0.56 | 4.6 |
| Pop | 0.19 | 0.61 | 1.4 |
| Jazz | 0.61 | 0.67 | 5.2 |
| Hip-Hop | 0.64 | 0.40 | 5.0 |
| EDM | 0.53 | 0.63 | 4.2 |
| Klassik | 0.07 | 0.96 | 0.6 |

Tabelle 4.1: Genre-gebundene Zufallsauswahl: Ergebnisse für korrekte Profile.

Wie in Tabelle 4.1 zu sehen ist, liefert genre-gebundene Zufallsauswahl überwiegend niedrige bis mittlere Präzision. Am höchsten liegen die Werte für Rock, Hip-Hop, Jazz und EDM, während Pop und Klassik sehr geringe Präzision aufweisen. Die Streaks sind kurz, und die Diversität hoch, da die Empfehlungen zufällig verteilt sind.

In Tabelle 4.2 ist zu erkennen, dass die Präzision in den meisten Genres ähnlich wie bei den korrekten Profilen ausfällt, mit Ausnahme von Pop, wo sie leicht zunimmt. Die Streaks bleiben kurz und die Diversität hoch, was den zufälligen Charakter der Empfehlungen widerspiegelt. Die Unterschiede zwischen korrekten und inkorrekten Profilen sind minimal.

| Genre | Präzision | Diversität | Streak |
|---|---|---|---|
| Rock | 0.66 | 0.64 | 4.8 |
| Pop | 0.38 | 0.63 | 3.0 |
| Jazz | 0.60 | 0.67 | 5.6 |
| Hip-Hop | 0.57 | 0.48 | 6.0 |
| EDM | 0.63 | 0.61 | 5.4 |
| Klassik | 0.11 | 0.99 | 1.2 |

Tabelle 4.2: Genre-gebundene Zufallsauswahl: Ergebnisse für inkorrekte Profile.

Die Balkendiagramme (siehe im Anhang A.6) zeigen, dass die mittleren Präzisionen nur durch die positiven Rückmeldungen erzielt wurden und kaum stark positiven Feedback erhalten. Auch ist die Leistung über verschiedenen Nutzern innerhalb des Genres sehr inkonsistent.

### 4.1.2 k-NN

| Genre | Präzision | Diversität | Streak |
|---|---|---|---|
| Rock | 0.84 | 0.15 | 9.8 |
| Pop | 0.49 | 0.20 | 4.2 |
| Jazz | 0.94 | 0.13 | 17.6 |
| Hip-Hop | 0.91 | 0.15 | 13.4 |
| EDM | 0.90 | 0.16 | 13.4 |
| Klassik | 0.99 | 0.10 | 19.6 |

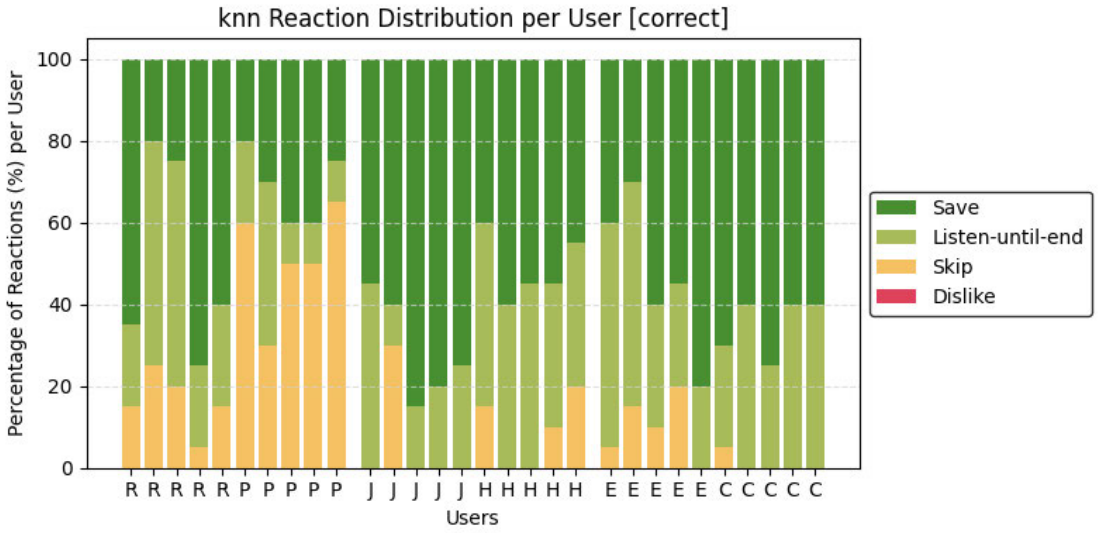
Tabelle 4.3: k-NN-Empfehlung: Mittelwert der Ergebnisse für korrekte Profile.

Wie in der Tabelle 4.3 zu erkennen, erreicht k-NN hohe Präzision und lange Streaks, insbesondere bei Jazz, Hip-Hop, EDM und Klassik. Rock zeigt ebenfalls hohe Präzision, jedoch etwas kürzere Streaks als Klassik. Pop ist das einzige Genre mit deutlich geringerer Präzision. Die Diversität ist generell niedrig, was sehr ähnliche Empfehlungen innerhalb eines Genres anzeigt.

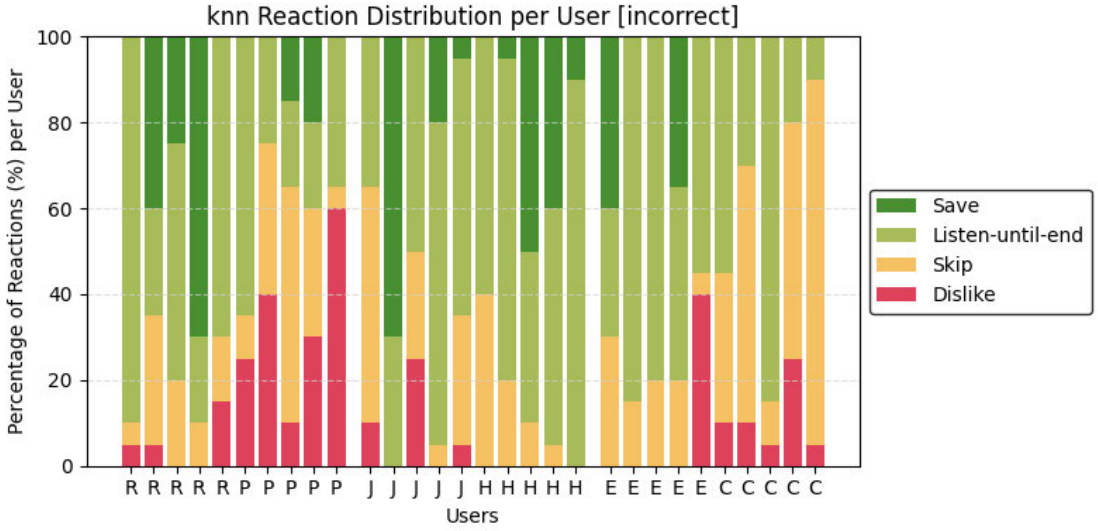
Bei inkorrekten Profilen (siehe Tabelle 4.4) fällt die Präzision in allen Genres ab, und die Streaks werden kürzer. Besonders Klassik profitiert stark von korrekten Profilen, während das Modell bei inkorrekten Profilen kaum noch treffsichere Empfehlungen liefert. Die Diversität steigt leicht in einigen Genres, zeigt aber weiterhin, dass die Empfehlungen relativ ähnlich bleiben.

| Genre | Präzision | Diversität | Streak |
|---|---|---|---|
| Rock | 0.79 | 0.29 | 9.8 |
| Pop | 0.40 | 0.27 | 3.0 |
| Jazz | 0.69 | 0.23 | 10.4 |
| Hip-Hop | 0.85 | 0.16 | 10.2 |
| EDM | 0.74 | 0.21 | 7.4 |
| Klassik | 0.40 | 0.09 | 4.2 |

Tabelle 4.4: k-NN-Empfehlung: Mittelwert Ergebnisse für inkorrekte Profile.




(a) Korrekte Profile




(b) Inkorrekte Profile

Abbildung 4.1: Vergleich der Nutzerreaktionen über 20 Empfehlungen für k-NN. Balken zeigen die Anteile der vier Feedbacktypen.

Die Feedbackverteilung zeigt bei korrekten Profilen (siehe Abbildung 4.1a) einen hohen Anteil von stark positiven über alle Genre mit langen Streaks. Auch wenn die Präzision für korrekte Profile bei Pop niedrig ist, hat keiner der Empfehlungen eine stark negative Reaktion verursacht.

Bei der Feedbackverteilung für inkorrekten Profilen (siehe Abbildung 4.1b) erkennt man einen starken Rückgang von stark positiven Reaktionen, besonderes bei Klassik. Während Nutzer mit Hip-Hop und EDM keine stark negativen Feedback abgegeben haben, ist dieser Teil besonders bei Pop und Klassik gestiegen.

### 4.1.3 DQN

| Genre | Präzision | Diversität | Streak |
|---|---|---|---|
| Rock | 0.76 | 0.47 | 5.4 |
| Pop | 0.68 | 0.57 | 6.4 |
| Jazz | 0.90 | 0.40 | 11.8 |
| Hip-Hop | 0.80 | 0.38 | 9.0 |
| EDM | 0.71 | 0.57 | 6.4 |
| Klassik | 0.50 | 0.73 | 6.0 |

Tabelle 4.5: DQN-Empfehlung: Mittelwert der Ergebnisse für korrekte Profile pro Genre.

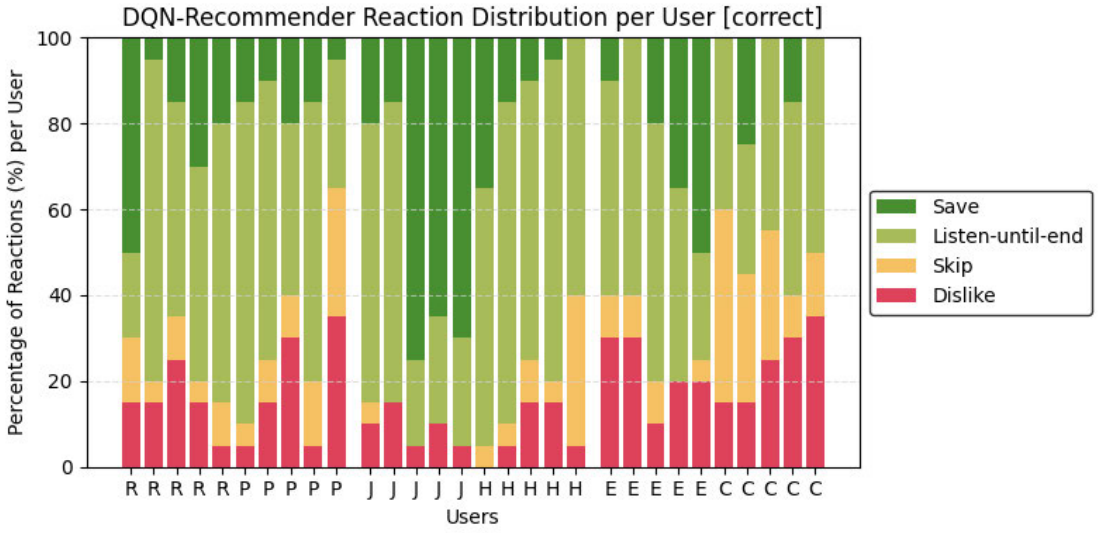
Nach Tabelle 4.5 hat DQN mittlere bis hohe Präzisionen über alle Genres erzielt. Besonders Nutzer mit Jazz und Hip-Hop fallen durch sehr hohe Präzisionen und lange Streaks auf. Pop und EDM erzielen mittlere Präzisionen bei gleichzeitig hoher Diversität. Klassik zeigt eher niedrige Präzision.

| Genre | Präzision | Diversität | Streak |
|---|---|---|---|
| Rock | 0.88 | 0.43 | 11.8 |
| Pop | 0.65 | 0.58 | 7.0 |
| Jazz | 0.87 | 0.38 | 9.4 |
| Hip-Hop | 0.91 | 0.34 | 14.0 |
| EDM | 0.72 | 0.59 | 7.6 |
| Klassik | 0.79 | 0.48 | 7.8 |

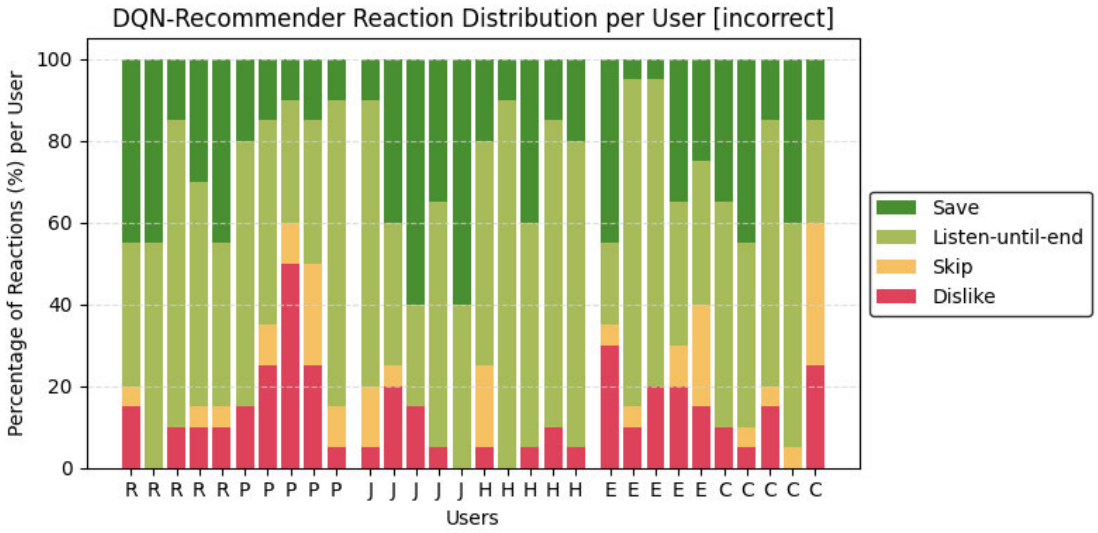
Tabelle 4.6: DQN-Empfehlung: Mittelwert der Ergebnisse für inkorrekte Profile.

Bei inkorrekten Profilen (siehe Tabelle 4.6) bleibt die Präzision in den meisten Genres stabil oder steigt sogar leicht an. Rock und Hip-Hop erreichen besonders lange Streaks.

Jazz verliert etwas Stabilität, bleibt aber auf einem hohen Präzisionsniveau. Klassik verbessert sich deutlich in Präzision, während die Diversität dort abnimmt.




(a) Korrekte Profile




(b) Inkorrekte Profile

Abbildung 4.2: Vergleich der Nutzerreaktionen über 20 Empfehlungen für DQN. Balken zeigen die Anteile der vier Feedbacktypen.

Die Feedbackverteilung von den korrekten Profilen (siehe Abbildung 4.2a) zeigt, dass das Modell für alle Nutzer hauptsächlich nur positive Reaktionen erbringt. Nur bei Jazz hat es für drei der fünf Nutzer geschafft einen großen Anteil an stark positiven Empfehlungen zu geben. Über fast alle Nutzer gibt das Modell stark negative Empfehlungen mit einem eher kleinen Anteil an nur negativen Empfehlungen.

Bei den inkorrekten Profilen (siehe Abbildung 4.2b) erkennt man einen Anstieg an stark positiven Empfehlungen über alle Genres. Trotzdem besteht ein Anteil an stark negativen Empfehlungen über fast allen Nutzern.

## 4.2 Ergebnisse im Vergleich

### 4.2.1 Präzision

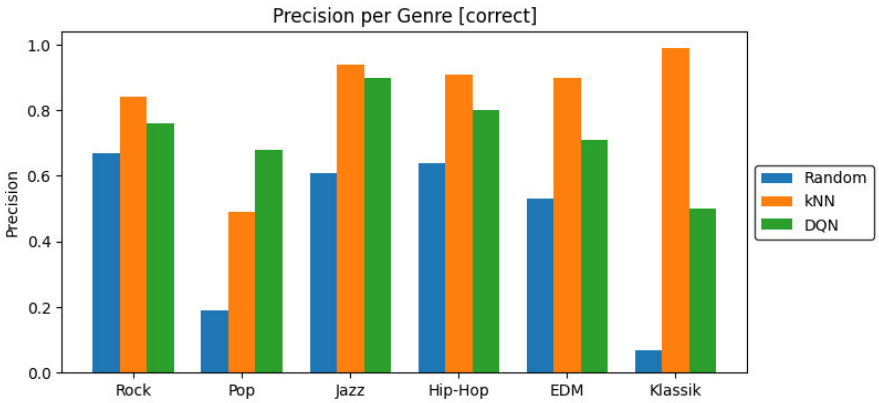



Abbildung 4.3: Balkendiagramm zeigt die erzielten durchschnittlichen Werte für Präzision pro Genre für alle drei Modelle.

Balkendiagramm 4.3 zeigt, dass k-NN in fast allen Genres die höchste Präzision erreicht, während DQN überwiegend mittlere bis hohe Werte liefert und Random deutlich schwächer ist. DQN übertrifft k-NN nur bei Pop, was auf eine bessere Anpassung an dieses Genre hindeutet.

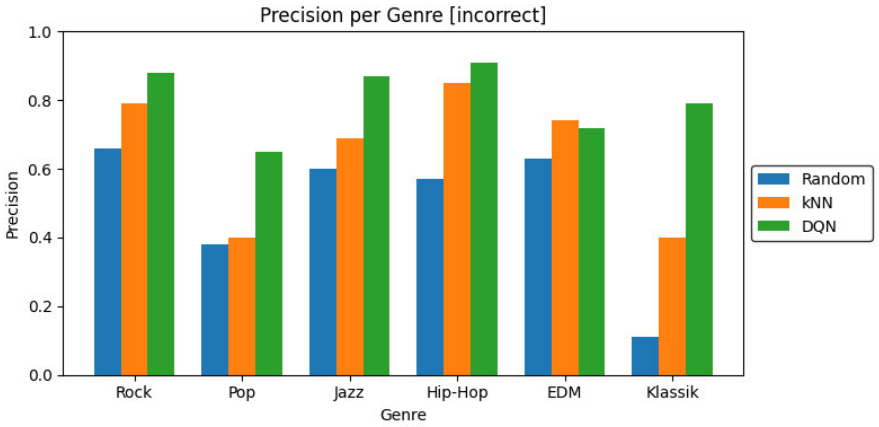



Abbildung 4.4: Balkendiagramm zeigt die erzielten durchschnittlichen Werte für Präzision pro Genre für alle drei Modelle.

Bei inkorrekten Profilen (siehe Abbildung 4.4) behält k-NN hohe Präzision in Jazz, Hip-Hop und EDM, während DQN die Präzision bei Pop und Klassik übertrifft. Random

bleibt konstant niedrig. Insgesamt zeigt DQN die stabilste Leistung gegenüber fehlerhaften Profilen.

### 4.2.2 Diversität

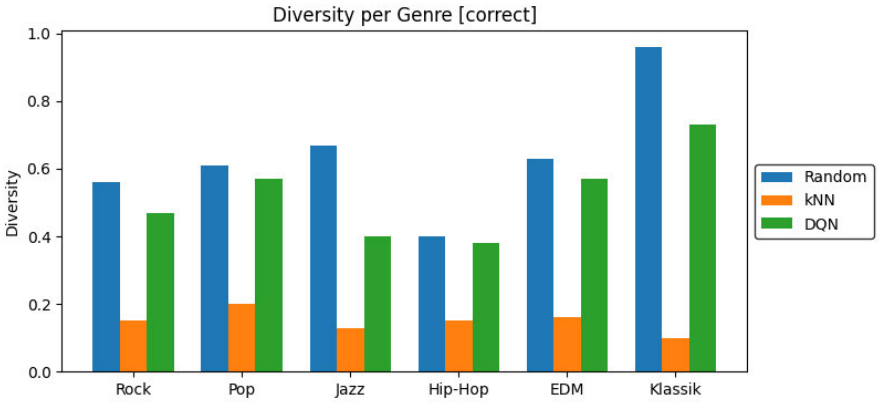



Abbildung 4.5: Balkendiagramm zeigt die erzielten durchschnittlichen Werte für Diversität pro Genre für alle drei Modelle.

Random erzeugt durchgehend die höchste Diversität, k-NN die niedrigste, während DQN dazwischen liegt (siehe Abbildung 4.5). Dies verdeutlicht, dass DQN einerseits auf Nutzerpräferenzen reagiert, andererseits eine gewisse Variation beibehält.

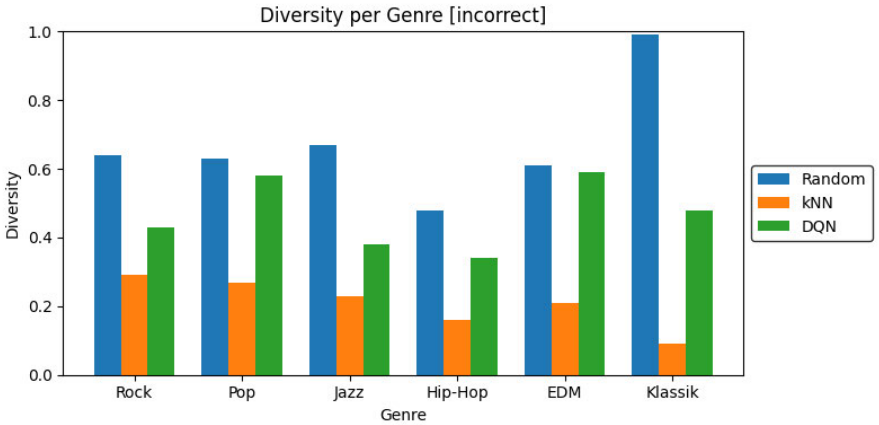



Abbildung 4.6: Balkendiagramm zeigt die erzielten durchschnittlichen Werte für Diversität pro Genre für alle drei Modelle.

Für inkorrekte Profile ergibt sich ein ähnliches Bild nach Abbildung 4.6.

### 4.2.3 Stabilität

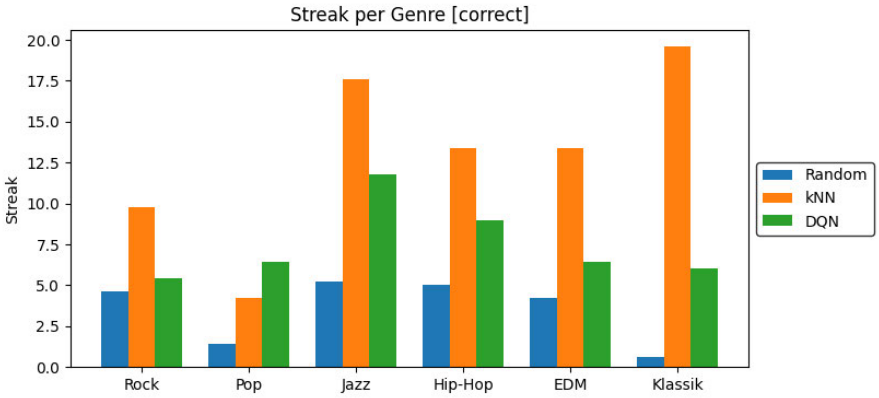



Abbildung 4.7: Balkendiagramm zeigt die erzielten durchschnittlichen Werte für Streak-Länge pro Genre für alle drei Modelle.

Zu Stabilität (siehe Abbildung 4.7)), erzielt k-NN die längsten zusammenhängenden positiven Empfehlungen. DQN liegt überwiegend im mittleren Bereich und Random hat kurze Streaks. Dies bestätigt die Stärke von k-NN in konsistenten Empfehlungen.

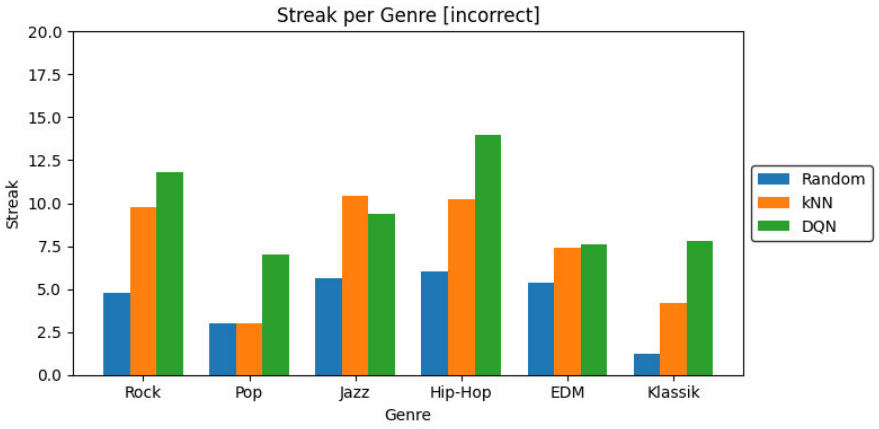



Abbildung 4.8: Balkendiagramm zeigt die erzielten durchschnittlichen Werte für Streak-Länge pro Genre für alle drei Modelle.

Bei inkorrekten Profilen (siehe Abbildung 4.8) zeigt DQN die längsten Streaks, besonders in Rock und Hip-Hop, während k-NN nur in einigen Genres noch moderate Streaks erreicht. Random bleibt schwach. DQN demonstriert hier seine Anpassungsfähigkeit.

### 4.2.4 Empfehlungsqualität

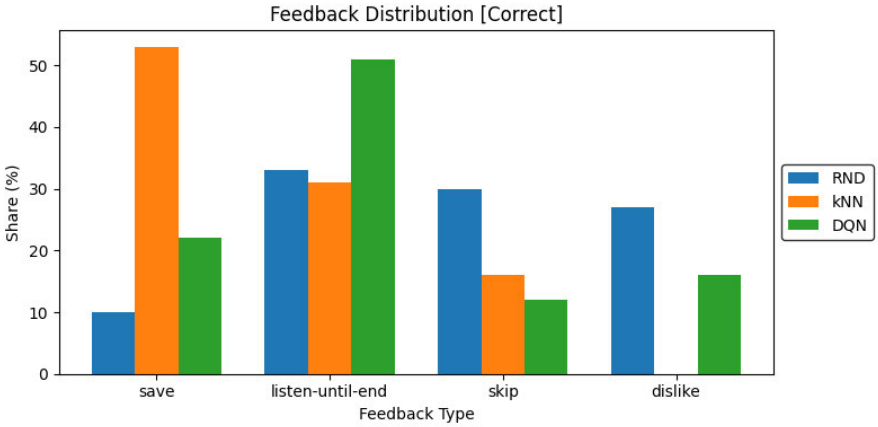



Abbildung 4.9: Balkendiagramm zeigt die Anteile der Reaktionen über alle Nutzer für alle drei Modelle.

Anhand Abbildung 4.9 erkennt man, dass k-NN erzeugt die meisten „save“-Reaktionen und keine „dislike“-Reaktionen, DQN liegt dazwischen, Random ist gleichmäßig verteilt. Das verdeutlicht die Balance zwischen Effektivität und Vielfalt bei DQN, aber zeigt, dass es im Vergleich zu k-NN zu unbeliebten Empfehlungen führt.

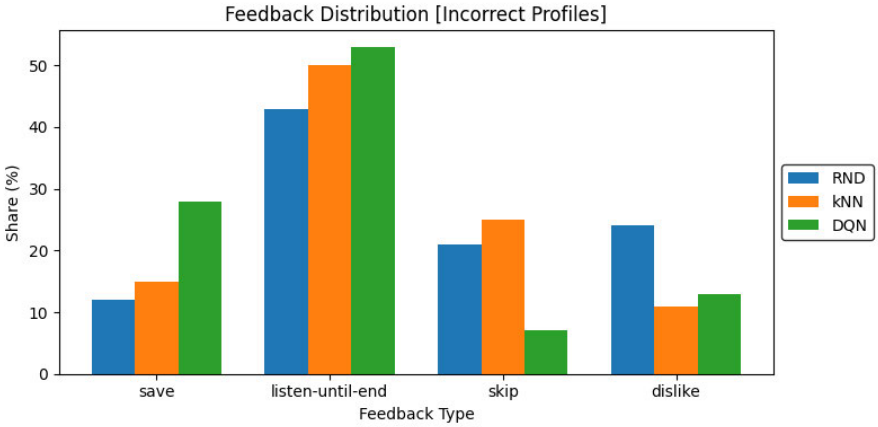



Abbildung 4.10: Balkendiagramm zeigt die Anteile der Reaktionen über alle Nutzer für Streak-Länge pro Genre für alle drei Modelle.

In Abbildung 4.10 zeigt, dass Random bleibt ausgeglichen. k-NN verliert deutlich an „save“ und steigt in „skip“ und „dislike“, während DQN stabile „listen“- und moderate

„save“-Anteile behält. DQN zeigt damit die größte Robustheit gegenüber inkorrekten Profilen.

# 5 Diskussion

## 5.1 Interpretation der Ergebnisse

Die Auswertung zeigt deutliche Unterschiede zwischen präferenz-treuen und explorierenden Nutzer. Die Genre-gebundene Zufallsauswahl zeigt, dass für diese Aufgabe die groben Genreeinordnungen nicht ausreichend ist, um qualitative Empfehlungen zu generieren. Präferenz-treue Nutzer erhalten nur selten positive Empfehlungen, da diese bei 200 Songs schwer zu treffen sind. Explorierende Nutzer schneiden auch nur besser ab, weil sie insgesamt mehr Songs im Genre positiv bewerten, doch das Ergebnis bleibt qualitativ weit hinter den anderen Methoden zurück.

k-NN erzielt bei zusammenhängenden Referenzsong und Hörverlauf die höchste Präzision und sehr lange Streaks, da klar Nachbarschaften zuverlässig erkannt werden. Die Feedbackqualität ist hier besonders auffällig, denn Nutzer mit dem Wunsch, ihren Interessen zu vertiefen, geben k-NN fast ausschließlich positive Rückmeldungen.

Bei explorierenden Nutzern sinkt die Präzision deutlich, bleibt jedoch in vielen Genres auf einem noch akzeptablen Niveau, bei dem die Mehrheit der Nutzer positive Bewertungen abgibt. Dies liegt daran, dass die Simulationsumgebung bereits empfohlene Songs aus dem Aktionsraum ausschließt. Dann teilen sich die Songs innerhalb ihre Genres noch teilweise relevante Merkmale, sodass selbst suboptimale Profile nicht automatisch zu schlechten Empfehlungen führen.

Besonders in Klassik kann man einen starken Abfall von Präzision und Feedbackqualität sehen, welche auf die Überspezialisierung von k-NN zurückzuführen ist. Wie man auch in der Abbildung A.2 erkennen kann, sind die Embeddings primär in zwei abgrenzbaren Clustern eingeteilt, aber sonst sehr dicht beieinander stehen. Bei explorierenden Nutzern zeigt sich, dass k-NN zunächst die Songs aus dem bekannten Cluster empfiehlt, selbst wenn ein Klassik-Nutzer eigentlich Songs aus dem anderen Cluster bevorzugen würde. Erst wenn die Songs im ursprünglichen Cluster weitgehend ausgeschöpft sind, schlägt

k-NN Empfehlungen aus dem anderen Cluster vor. Dies verdeutlicht, dass k-NN stark an den vorhandenen Ähnlichkeiten festhält.

Nur die Nutzer mit Pop-Historien zeigen, dass k-NN abgesehen von den Nutzertypen eine gleiche Leistung vorzeigt. Popsongs teilen oft Merkmale mit anderen Genres und zeichnen sich als eigenes Genre nur durch ihre Popularität aus. Entsprechend sind die Song-Embeddings sehr unterschiedlich zueinander, wodurch k-NN sich schwerer durch die Nachbarschaft orientieren kann und oft Songs von anderen Genres auch empfehlt, die ähnlich klingen. Damit ist k-NN besonders geeignet für Szenarien, in denen präzise Nutzerdaten vorliegen, aber empfindlich gegenüber ungenaue oder verrauschte Profilen.

DQN hingegen zeigt eine größere Flexibilität. Bei präferenz-treuen Nutzern profitieren die Empfehlungen eher weniger von der Diversität der Empfehlungen. Das Modell liefert überwiegend positive Empfehlungen, allerdings immer mit einem kleinen, konstanten Anteil an stark negativen Bewertungen. Weil DQN explorativer vorgeht, braucht es eine kurze Lernphase, um sich vom Feedback des Nutzers leiten zu lassen. Besonders am Anfang werden dann unpassende Titel auswählt bevor es ein richtiges Muster erkennt.

Bei explorierenden Nutzern und bei Pop, wo k-NN durch ihre niedrige Diversität an Präzision verliert, erreicht DQN in einigen Genres längere Streaks und fast immer höhere Präzision. Da profitieren die Nutzer von den am Anfang diversen Empfehlungen und geben im Laufe der Empfehlungssession genug Feedback-Signale zurück, sodass DQN für die ein Muster die weiteren Empfehlungen findet.

Besonders bei den explorierenden Nutzern, die Klassik hören, fehlt der konstante Anteil an stark negativen Bewertungen und auch bei Hip-Hop Nutzern ist der Anteil minimal im Vergleich zu den präferenz-treuen Nutzern. Das zeigt, dass die Diversität am Anfang der Session nur von Vorteil ist, wenn es klar ist, dass die aktuellen Nutzer von ihren aktuellen Präferenzen explorieren wollen. Die Feedbackqualität bleibt bei DQN jedoch gemischt, vorwiegend nur positive und im Vergleich zu k-NN wenigen stark positiven Treffer, aber auch mit vereinzelten stark negativen Fehlgriffen.

Bei Genres wie Rock, Hip-Hop und EDM zeigt sich, dass beide Modelle grundsätzlich solide arbeiten, aber mit unterschiedlichem Fokus. k-NN überzeugt durch hohe Präzision und langen Streaks, wenn der Nutzer seine Präferenzen vertiefen möchte, während DQN bei explorierenden Nutzern die gleiche bzw. bessere Ergebnisse liefert. Besonders bei Klassik, wo mittlere und hohe Ähnlichkeit dominieren, kann DQN durch Feedbackorientierung

sich schnell zu den optimalen Empfehlungen anpassen, während k-NN dazu neigt, in eng begrenzten Nachbarschaften zu verharren.

Insgesamt zeigt sich, dass k-NN bei Nutzern, die sich in ihre Präferenz vertiefen wollen, am stärksten ist, mit sehr hoher Feedbackqualität. DQN ist dafür robuster gegenüber schwer inakkurate Profilabbildungen und kann breitere Präferenzmuster bedienen. Zudem erreicht DQN selbst bei präferenz-treuen Nutzern nicht immer die Spitzenpräzision von k-NN. Außerdem schwankt die Feedbackqualität durch das explorative Verhalten, denn DQN setzt eher einen Fokus auf die ausreichend positiven Bewertungen als die stark positiven.

## 5.2 Praktische Relevanz

Die Ergebnisse zeigen, dass k-NN bei präferenz-treuen Nutzern eine hohe Präzision und konsistente Empfehlungen für bekannte Musikpräferenzen liefert. DQN kann sich durch Feedbacksignale im Embedding-Raum orientieren und erzielt ebenfalls hohe Präzisionen, allerdings auf Kosten der Qualität einzelner Empfehlungen. Der Vorteil gegenüber k-NN ist Genre abhängig und nur teilweise erkennbar.

Besonders bei explorierende Nutzern zeigt DQN eine robustere Leistung, indem es länger zusammenhängende positiv bewertete Empfehlungen generiert, während k-NN hier deutliche Einbußen verzeichnet. Insgesamt liefert k-NN jedoch bereits in beiden Szenarien brauchbare Empfehlungen, ohne den hohen Implementierung- und Trainingsaufwand eines RL-Agenten. Die Stärken von DQN liegen vor allem in explorativen Empfehlungen und der Anpassungsfähigkeit an größere Nutzer- und Songvielfalt, wodurch Nutzer stärker in den Empfehlungsprozess einbezogen werden können. In der vereinfachten Simulationsumgebung zielt DQN auf langfristige Belohnung ab, was suboptimale Empfehlungen in Kauf nimmt, um zukünftige positive Reaktionen zu maximieren. Für den alltäglichen Gebrauch, bei dem Nutzer überwiegend ihrem bisherigen Geschmack folgen, bleibt k-NN die stabilere und effizientere Methode.

## 5.3 Limitationen

Die Ergebnisse dieser Arbeit müssen im Lichte mehrere Einschränkungen betrachtet werden. Zunächst basiert die Evaluation auf synthetisch generierte Nutzerdaten mit

Heuristiken, die nicht empirisch validiert sind. Dadurch fehlt der direkte Bezug zu realem Nutzerverhalten, wodurch die Resultate nur relativ zur Simulationsumgebung interpretiert werden können.

Die begrenzte Größe des verwendeten Songdatensatzes mit rund 1200 Titeln schränkte vor allem die Anzahl der möglichen Nutzerprofile ein und wirkt sich somit auf die Variation der Simulationsumgebungen aus. Die Selbst-Umsetzung der Nutzersimulation stellt ebenfalls eine Einschränkung dar, weil Kontrolle und Monitoring teilweise fehlen. Zwar wird das Nutzerfeedback deterministisch generiert, jedoch basiert es auf lokalen Ähnlichkeiten, was die Nachvollziehbarkeit einzelner Entscheidungen erschwert. Zudem fehlt eine statistische Absicherung der Parameterwahl, die das Verhalten der Nutzer besser interpretierbar machen würde.

Da die Parameter global bei der Erstellung der Nutzer gesetzt wurden, haben Nutzer unterschiedlich große Mengen an relevanten Songs. Session-basierte Effekte und Überschneidungen der Hörhistorien einzelner Nutzer wurden nur teilweise berücksichtigt. Außerdem kann es vorkommen, dass mehrere Nutzer dieselben relevanten Songs besitzen, diese aber in unterschiedlicher Reihenfolge hören. Schließlich kann die Anzahl wirklich relevanter Songs bei Genres mit großer Spanne in Erscheinungsjahr oder Länge eingeschränkt sein.

Auch die Nutzersimulation beschränkt sich auf nur einzelne Genrepräferenzen und reduziert die Präferenzstruktur auf binäre Szenarien. Es werden keine Mischinteressen oder verändernde Vorlieben berücksichtigt, sodass keine realistischen Dynamiken über längere Zeiträume untersucht werden können.

Schließlich ist auch die Evaluation der Empfehlungsqualität selbst vereinfacht durch das enge Szenariosetting und dem Design des Feedbackmechanismus. Die Fokussierung auf kurze Sessions erlaubt keine Aussage zu langfristigen Nutzerbindung. Zudem basieren die Experimente auf einer kleinen Anzahl synthetischer Nutzer (60 insgesamt) und wurden nur über einen Seed durchgeführt. Das verstärkt statistische Schwankungen und kann die Stabilität der Ergebnisse beeinflussen.

Zusammenfassen schränken die vereinfacht die reduzierte Simulationsumgebung, die eingeschränkte Methodenauswahl sowie die Evaluation auf wenige Metriken die Generalisierbarkeit der Ergebnisse ein. Die vorliegende Arbeit ermöglicht trotzdem einen systematischen Vergleich der gewählten Ansätze, sollt aber primär als explorative Arbeit verstanden werden, die die Ansatzpunkte für weiterführende Forschung bietet.

## 5.4 Ausblick

Die vorliegende Ergebnisse basieren auf dem Vergleich jeweils einer sehr grundlegenden Methode der klassischen ML-Methoden und Reinforcement Learning. Sie zeigt, dass RL einen Mehrwert für Musikempfehlungssysteme bringt, besonders zur Unterstützung von explorativen Verhalten der Nutzer. Diese sind jedoch eher minimal gehalten, welche sich auch auf die Einschränkungen der ausgewählten Algorithmen bezieht. Daher liefern die Resultate nur einen eingeschränkten Einblick in die Leistungsfähigkeit moderner Musikempfehlungssysteme und es besteht noch deutliches Potenzial für Verbesserungen.

Zukünftige Arbeiten können verschiedene Richtungen verfolgen, um den Vergleich realistischer und aussagekräftiger zu gestalten.

Ein erweiterter Aktionsraum für das RL-Modell ließe sich durch Verfahren realisieren, die kontinuierliche Aktionen verarbeiten können. Denkbar wäre hier der Einsatz eines Actor-Critic-Ansatzes [16] oder eines hybriden Modells auf Basis von On-Policy Proximal Policy Optimization [17].

Darüber hinaus könnte die Modellierung der Songs verbessert werden. In der vorliegenden Arbeit wurde die Cosine-Similarity zur Bestimmung der Ähnlichkeit verwendet, die nur lokale Nachbarschaften berücksichtigt. Alternativ könnten komplexere Item-Merkmale oder Embeddings eingesetzt werden, die globale Strukturen und musikspezifische Merkmale besser abbilden.

Zusätzlich wäre eine realistischere Nutzersimulation denkbar, um die Varianz menschlichen Feedbacks zu berücksichtigen. Auch die Berücksichtigung von Langzeiteffekten, Metadaten und mehrfache Sessions pro Nutzer könnte die Stabilität der Ergebnisse erhöhen und die Lernverläufe realistischer machen.

Insgesamt zeigen diese offenen Punkte und Erweiterungsmöglichkeiten, dass die vorliegende Arbeit eine erst Grundlage für die Untersuchung von Reinforcement Learning-basierten Musikempfehlungssystem bietet. Es bleibt außerdem offen, inwiefern die hier gewonnen Ergebnisse auf reale Nutzer und größere, komplexere Musikbibliotheken übertragbar sind, oder ob die vereinfachte Simulationsumgebung und die begrenzte Komplexität der Modelle die Resultate beeinflusst haben.

# Literaturverzeichnis


[1] Kowald, Dominik ; Lacic, Emanuel: Popularity Bias in Collaborative Filtering-Based Multimedia Recommender Systems. https://arxiv.org/abs/2203.00376. Version: 2022

[2] Mok, Lillio ; Way, Samuel F. ; Maystre, Lucas ; Anderson, Ashton: The Dynamics of Exploration on Spotify. In: Proceedings of the International AAAI Conference on Web and Social Media 16 (2022), May, 663-674. http://dx.doi.org/10.1609/icwsm.v16i1.19324. – DOI 10.1609/icwsm.v16i1.19324

[3] Tomasi, Federico ; Cauteruccio, Joseph ; Kanoria, Surya ; Ciosek, Kamil ; Rinaldi, Matteo ; Dai, Zhenwen: Automatic Music Playlist Generation via Simulation-based Reinforcement Learning. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York, NY, USA : Association for Computing Machinery, 2023 (KDD '23). – ISBN 9798400701030, 4948–4957

[4] Liebman, Elad ; Saar-Tsechansky, Maytal ; Stone, Peter: DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. https://arxiv.org/abs/1401.1880. Version: 2015

[5] Chen, Boyuan ; Zhu, Chuning ; Agrawal, Pulkit ; Zhang, Kaiqing ; Gupta, Abhishek: Self-Supervised Reinforcement Learning that Transfers using Random Features. https://arxiv.org/abs/2305.17250. Version: 2023

[6] Kapitel 1. In: Ricci, Francesco ; Rokach, Lior ; Shapira, Bracha: Introduction to Recommender Systems Handbook. Boston, MA : Springer US, 2011. – ISBN 978–0–387–85820–3, 1-35

[7] Gunawardana, Asela ; Shani, Guy: A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. In: Journal of Machine Learning Research 10 (2009), Nr. 100, 2935–2962. http://jmlr.org/papers/v10/gunawardana09a.html

[8] Chemeque Rabel, Marine: Content-based music recommendation system: A comparison of supervised Machine Learning models and music features, KTH Royal Institute of Technology, Diss., 2020. https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-288534

[9] Jadon, Aryan ; Patil, Avinash: A Comprehensive Survey of Evaluation Techniques for Recommendation Systems. https://arxiv.org/abs/2312.16015. Version: 2024

[10] Schedl, Markus: Deep Learning in Music Recommendation Systems. In: Frontiers in Applied Mathematics and Statistics Volume 5 - 2019 (2019). http://dx.doi.org/10.3389/fams.2019.00044. – DOI 10.3389/fams.2019.00044. – ISSN 2297–4687

[11] Sutton, Richard S. ; Barto, Andrew G.: Reinforcement Learning: An Introduction. Cambridge, MA, USA : A Bradford Book, 2018. – ISBN 0262039249

[12] Cramer, Jason ; Wu, Ho-Hsiang ; Salamon, Justin ; Bello, Juan P.: Look, Listen, and Learn More: Design Choices for Deep Audio Embeddings. In: ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2019), 3852-3856. https://api.semanticscholar.org/CorpusID:146028516

[13] McFee, Brian ; Raffel, Colin ; Liang, Dawen ; Ellis, Daniel P. ; McVicar, Matt ; Battenberg, Eric ; Nieto, Oriol: librosa: Audio and music signal analysis in python. In: Proceedings of the 14th python in science conference Bd. 8, 2015

[14] Paszke, Adam ; Gross, Sam ; Massa, Francisco ; Lerer, Adam ; Bradbury, James ; Chanan, Gregory ; Killeen, Trevor ; Lin, Zeming ; Gimelshein, Natalia ; Antiga, Luca ; Desmaison, Alban ; Köpf, Andreas ; Yang, Edward ; DeVito, Zach ; Raison, Martin ; Tejani, Alykhan ; Chilamkurthy, Sasank ; Steiner, Benoit ; Fang, Lu ; Bai, Junjie ; Chintala, Soumith: PyTorch: An Imperative Style, High-Performance Deep Learning Library. https://arxiv.org/abs/1912.01703. Version: 2019

[15] Kunaver, Matevž ; Požrl, Tomaž: Diversity in recommender systems – A survey. In: Knowledge-Based Systems 123 (2017), 154-162. http://dx.doi.org/https://doi.org/10.1016/j.knosys.2017.02.009. – DOI https://doi.org/10.1016/j.knosys.2017.02.009. – ISSN 0950–7051

[16] Chen, Minmin ; Xu, Can ; Gatto, Vince ; Jain, Devanshu ; Kumar, Aviral ; Chi, Ed: Off-Policy Actor-critic for Recommender Systems. In: Proceedings of the 16th ACM Conference on Recommender Systems. New York, NY, USA : Association for Computing Machinery, 2022 (RecSys '22). – ISBN 9781450392785, 338–349

[17] Feng, Qian ; Xiao, Geyang ; Liang, Yuan ; Zhang, Huifeng ; Yan, Linlin ; Yi, Xiaoyu: Proximal Policy Optimization for Explainable Recommended Systems. In: 2022 4th International Conference on Data-driven Optimization of Complex Systems (DOCS), 2022, S. 1–6

# A Anhang

| Genre | Jahr min | Jahr max | Jahr avg | Jahr std |
|---|---|---|---|---|
| classical | 1972 | 2025 | 2010.73 | 13.71 |
| edm | 2001 | 2025 | 2022.92 | 2.85 |
| hip-hop | 1993 | 2025 | 2016.36 | 7.84 |
| jazz | 1954 | 2025 | 2004.52 | 25.54 |
| pop | 1980 | 2025 | 2016.13 | 7.94 |
| rock | 1964 | 2024 | 1993.82 | 13.90 |

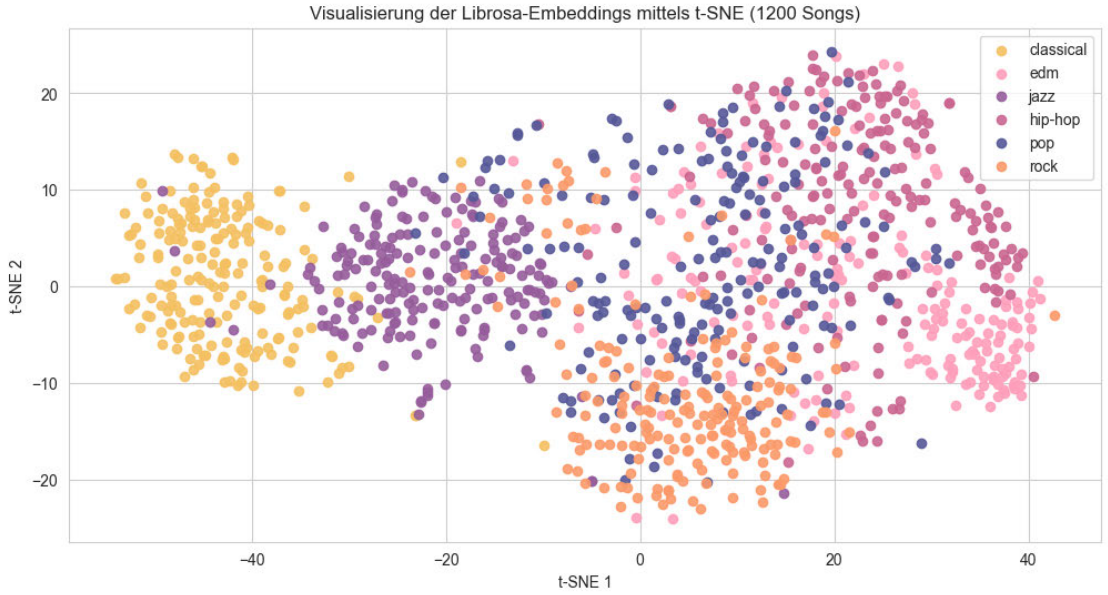
Tabelle A.1: Jahresstatistik der Songs pro Genre

| Genre | Länge min | Länge max | Länge mean | Länge std |
|---|---|---|---|---|
| classical | 1:21 | 27:17 | 4:47 | 3:10 |
| edm | 1:42 | 9:02 | 3:24 | 1:01 |
| hip-hop | 1:33 | 6:44 | 3:24 | 0:58 |
| jazz | 1:44 | 11:01 | 4:26 | 1:28 |
| pop | 1:44 | 5:50 | 3:29 | 0:42 |
| rock | 2:01 | 9:07 | 4:09 | 1:10 |

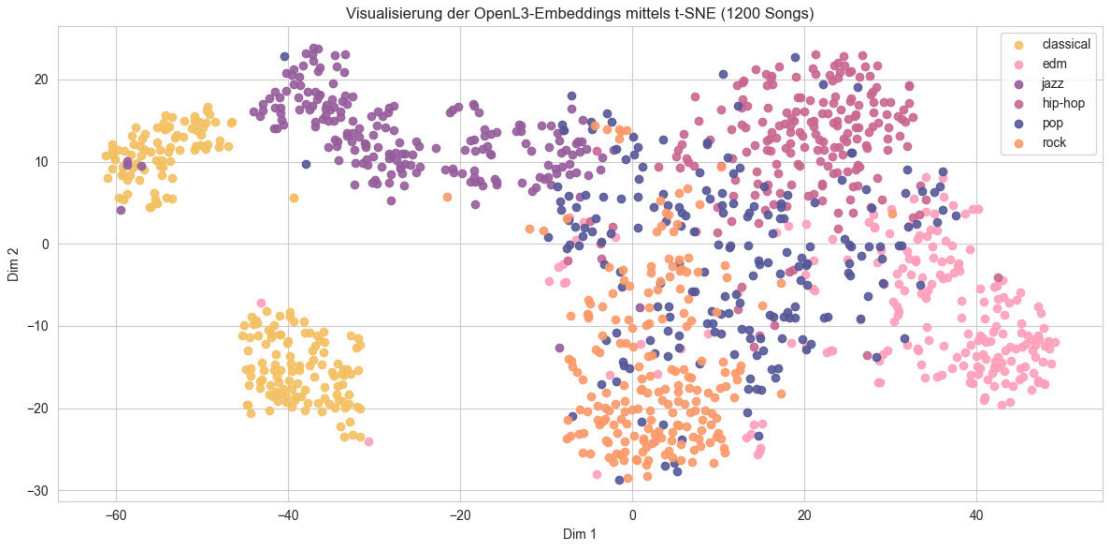
Tabelle A.2: Längenstatistik der Songs pro Genre

| Genre | Pop min | Pop max | Pop mean | Pop std | #Künstler |
|---|---|---|---|---|---|
| classical | 30 | 78 | 52.54 | 8.24 | 115 |
| edm | 17 | 87 | 55.32 | 18.46 | 144 |
| hip-hop | 36 | 94 | 71.69 | 9.69 | 131 |
| jazz | 18 | 63 | 46.73 | 6.12 | 152 |
| pop | 31 | 96 | 71.34 | 12.87 | 158 |
| rock | 33 | 93 | 76.73 | 9.35 | 133 |

Tabelle A.3: Popularitätsstatistik der Songs und Anzahl an distinkte Künstler pro Genre




(a) t-SNE der Librosa-Features




(b) t-SNE der OpenL3-Embeddings

Abbildung A.1: t-SNE Visualisierung der Features von 1200 Songs: Vergleich Librosa vs. OpenL3

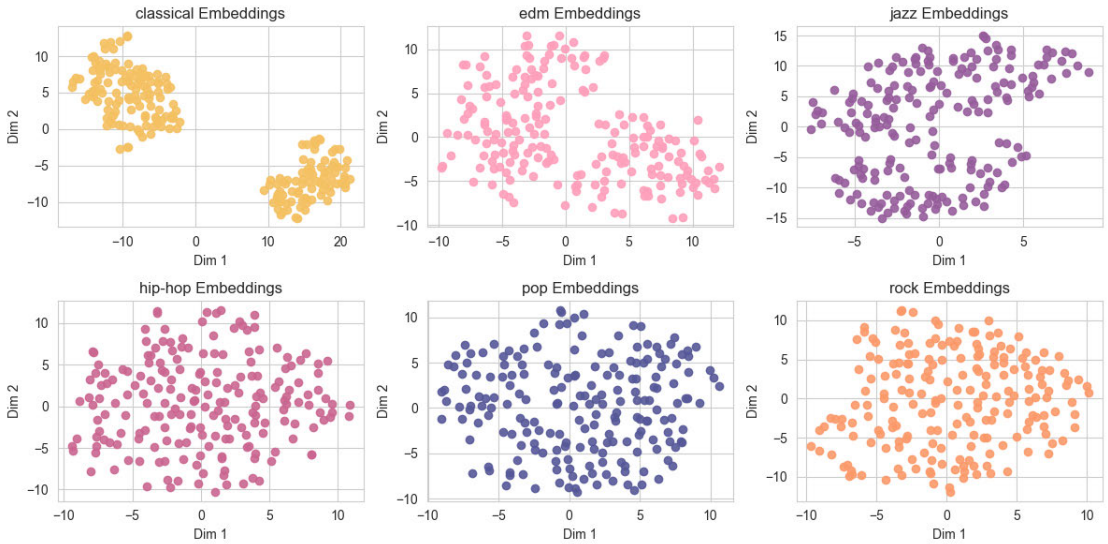



Abbildung A.2: t-SNE Visualisierung der OpenL3 Embeddings pro Genre

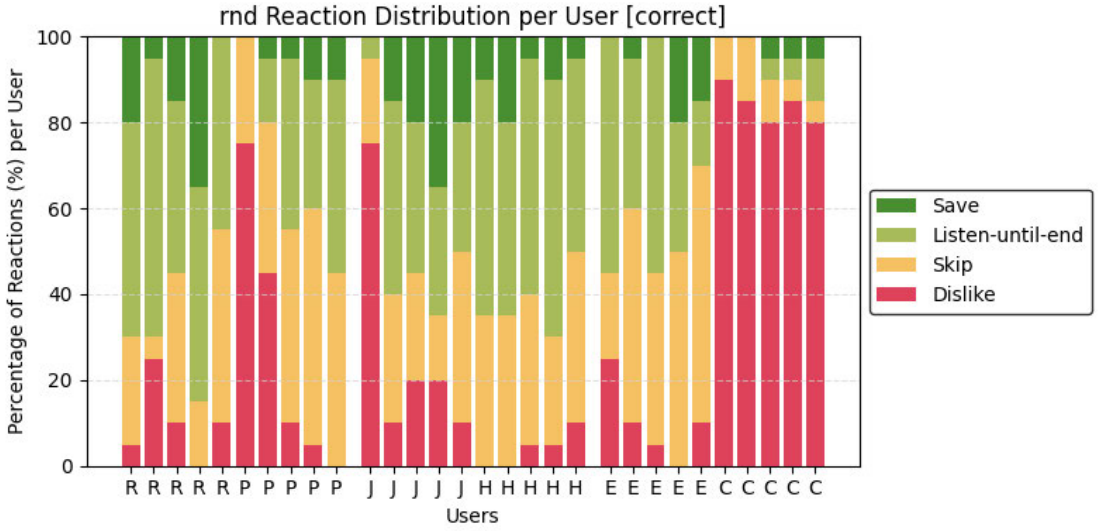
Abbildung A.3: Available songs per user (correct)

| Genre | User | #Pos. Songs |
|---|---|---|
| rock | 214 | 24 |
| rock | 126 | 48 |
| rock | 201 | 54 |
| rock | 56 | 58 |
| rock | 242 | 56 |
| pop | 148 | 31 |
| pop | 75 | 22 |
| pop | 97 | 24 |
| pop | 19 | 25 |
| pop | 133 | 23 |
| jazz | 177 | 35 |
| jazz | 16 | 28 |
| jazz | 227 | 67 |
| jazz | 45 | 65 |
| jazz | 106 | 57 |
| hip-hop | 179 | 48 |
| hip-hop | 182 | 54 |
| hip-hop | 195 | 52 |
| hip-hop | 251 | 67 |
| hip-hop | 82 | 55 |
| edm | 119 | 29 |
| edm | 102 | 37 |
| edm | 37 | 59 |
| edm | 138 | 70 |
| edm | 113 | 25 |
| classical | 101 | 67 |
| classical | 240 | 46 |
| classical | 91 | 57 |
| classical | 26 | 67 |
| classical | 29 | 64 |

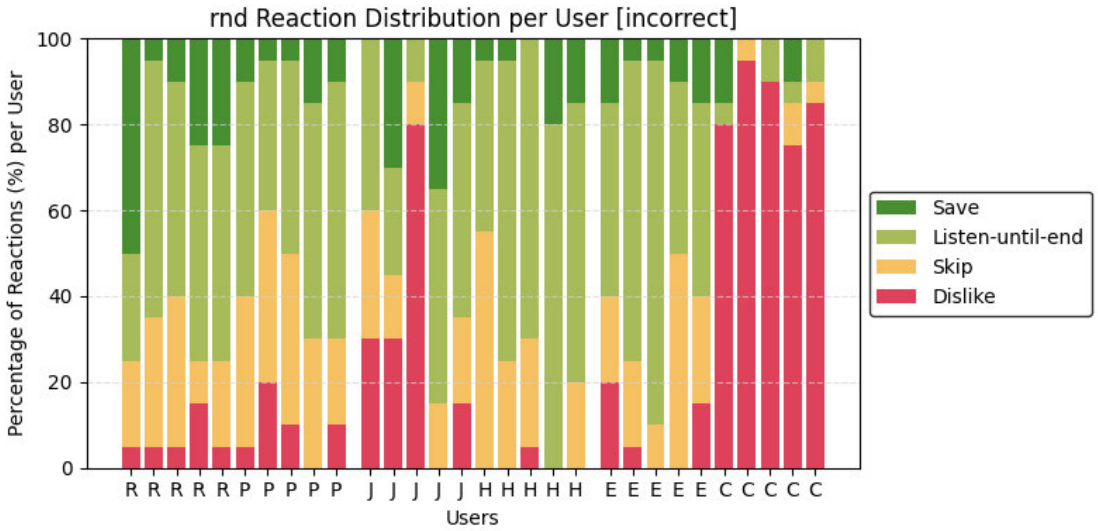
Abbildung A.4: Available songs per user (incorrect)

| Genre | User | #Pos. Songs |
|---|---|---|
| rock | 40 | 52 |
| rock | 170 | 62 |
| rock | 32 | 45 |
| rock | 168 | 53 |
| rock | 117 | 61 |
| pop | 63 | 30 |
| pop | 211 | 39 |
| pop | 38 | 26 |
| pop | 118 | 26 |
| pop | 78 | 37 |
| jazz | 193 | 46 |
| jazz | 183 | 26 |
| jazz | 230 | 59 |
| jazz | 254 | 63 |
| jazz | 111 | 61 |
| hip-hop | 210 | 41 |
| hip-hop | 55 | 39 |
| hip-hop | 100 | 21 |
| hip-hop | 208 | 67 |
| hip-hop | 21 | 58 |
| edm | 238 | 35 |
| edm | 137 | 67 |
| edm | 181 | 66 |
| edm | 79 | 66 |
| edm | 64 | 58 |
| classical | 151 | 67 |
| classical | 229 | 51 |
| classical | 92 | 62 |
| classical | 224 | 68 |
| classical | 135 | 67 |

Abbildung A.5: Übersicht der Testnutzer mit ihrer möglichen Anzahl an positiven Empfehlungen, sortiert nach Genre




(a) Korrekte Profile (30 Nutzer, 5 pro Genre). Buchstaben auf der X-Achse: R = Rock, P = Pop, J = Jazz, H = Hip-Hop, E = EDM, C = Classical.




(b) Inkorrekte Profile (30 Nutzer, 5 pro Genre). Buchstaben auf der X-Achse: R = Rock, P = Pop, J = Jazz, H = Hip-Hop, E = EDM, C = Classical.

Abbildung A.6: Vergleich der Nutzerreaktionen über 20 Empfehlungen. Balken zeigen die Anteile der vier Feedbacktypen.

## A.1 Verwendete Hilfsmittel

In der Tabelle A.4 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.4: Verwendete Hilfsmittel und Werkzeuge

| Tool | Verwendung |
|---|---|
| LaTeX | Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments |
| draw.io | Flowchart Maker und Online Diagram Software verwendet zur Erstellung von Grafiken |

## Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

| Ort | Datum | Unterschrift im Original |
|---|---|---|