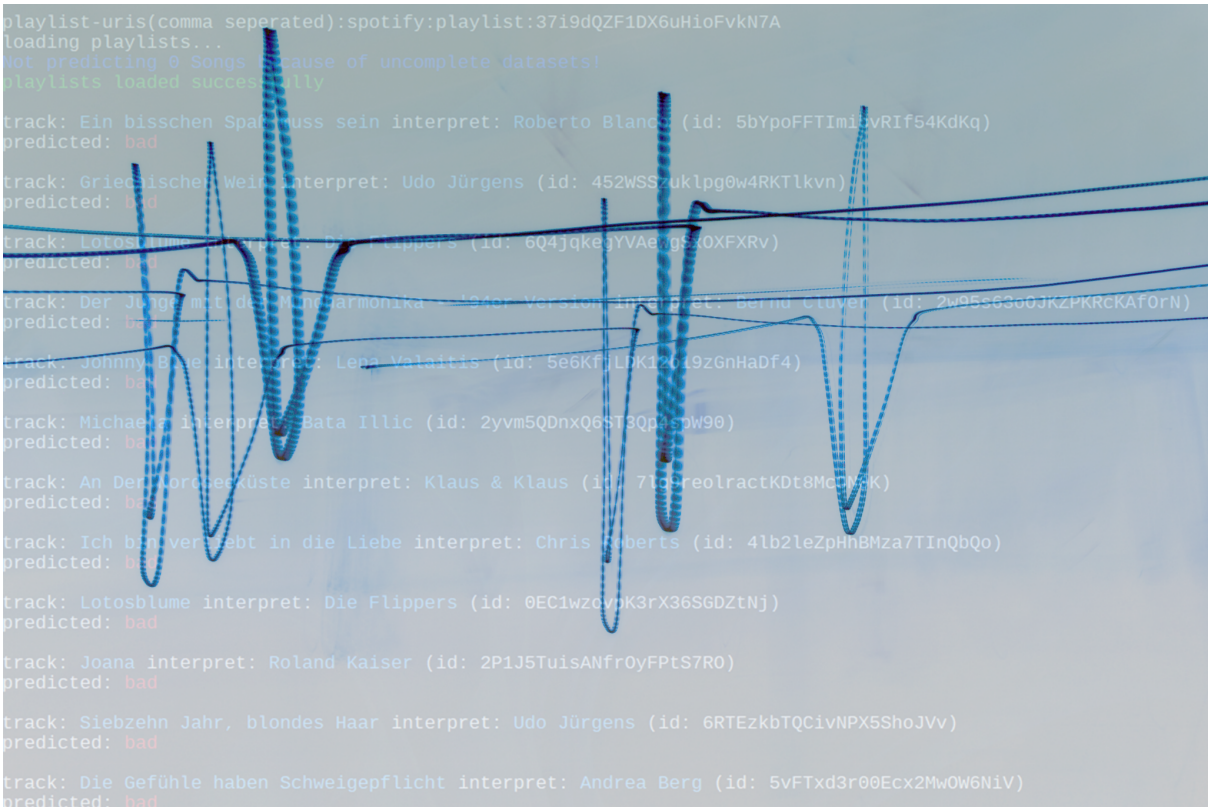
17.07.2021

Ziehenschule

Eine KI mit gutem Musikgeschmack

Informatik Facharbeit



Autor:	Felician Schwarz
Kurs:	Informatik GK Q3
Lehrkraft:	Herr Nguyen

Inhaltsverzeichnis

Einleitungki **3**

Vorbereitung und Erste Entwürfe **5**

Künstliche Intelligenz 5

Spotify als Datenquelle 5

Von Songs auf Spotify zu Daten für die KI 6

Implementierung **8**

Programmstruktur 8

spotify.py 9

main.py 9

Der Algorithmus „Support Vector Machine” **10**

Anwendung in der Praxis **12**

Bedienung des Prototyps 12

Ein Testdurchlauf 12

Auswertung der Ergebnisse & Ausblick für die Zukunft **16**

Fazit **17**

Anhang **18**

Quellen **27**

Eigenständigkeitserklärung **28**

Einleitung

In den Winterferien 2020/2021 habe ich den Prototyp einer künstlichen Intelligenz entwickelt, welche Musikgeschmäcker erlernen kann. Das entwickelte Computerprogramm kann auf Basis des Gelernten entscheiden, ob ausgewählte Lieder dem Nutzer gefallen oder nicht gefallen würden.
Die Entwicklung des Programms sollte hierbei nur etwa drei Tage in Anspruch nehmen und für mich vor allem eine Frage beantworten: „Kann eine KI intelligent genug sein, um eine so komplexe und menschliche Eigenschaft, wie einen Musikgeschmack zu erlernen?"
Des Weiteren sollte das Projekt eine neue Herausforderung meiner Programmierfähigkeiten sein, aber auch der Gedanke daran, den Vorgang der Suche nach neuer Musik, mit einem solchen Programm automatisieren zu können, faszinierte mich.

Ich habe mich schon immer sehr für Musik interessiert und habe auch in meiner Kindheit mehrere Instrumente gespielt. Schon seit Jahren höre ich täglich etwa ein- bis zwei Stunden Musik auf Spotify und habe dadurch auch schon riesige Mengen an Daten gesammelt, welche später noch eine wichtige Rolle spielen werden.
Grundsätzlich interessiere ich mich für viele verschiedene Musikgenres, wobei ich aber besonders häufig Musik aus dem Hip-Hop/ Rap Genre höre. Besonders die sogenannten „Beats" der Lieder hatten immer besondere Relevanz für mich. Unter Beats versteht man die meist digital erzeugten Instrumente samt Schlagzeug, welche den eigentlichen Rap, also den „Sprechgesang", begleiten. So habe ich in der Vergangenheit ebenfalls viele Beats selber produziert.
Nachdem ich also jahrelang sehr viel über Musik und über meinen eigenen Musikgeschmack gelernt habe, ist nichtsdestotrotz die Suche nach neuen Songs meist ein langer Weg, obwohl ich genau weiß, nach welchen Songs bzw. Beats ich suche, da ich meinen eigenen Geschmack bereits genaustens analysiert habe. Es blieb mir schlichtweg nichts anderes übrig, als lange Hip-Hop/ Rap Playlisten nach guten Songs zu durchsuchen. Hier kommt nun meine KI ins Spiel, welche mir diesen Prozess automatisieren sollte. Als Grundlage dafür sollten meine erstellten Playlisten dienen, in welchen ich über die Jahre tausende Lieder abgespeichert habe, weil sie mir offensichtlich sehr gut gefallen haben.
Das zu entwickelnde Programm sollte aber keinesfalls dieselben Ziele, wie die bereits vorhandenen Technologien für Song-Empfehlungen auf Basis der Nutzerdaten, welche viele Audio-Streaming-Dienste, wie auch Spotify, anbieten, verfolgen. Aus eigener Erfahrung vermute ich, dass diese Empfehlungen größtenteils darauf basieren, was andere Nutzer mit ähnlichen Daten auf der Plattform konsumieren, sowie auf dem Genre und den Künstlern. Mir geht es jedoch allein um die musikalische Gestalt bzw. den Charakter der Lieder, wobei ich mir daraus auch bessere Ergebnisse als durch Empfehlungen erhoffe. Die KI sollte möglichst nur Lieder auswählen, welche dem Nutzer auch wirklich gefallen, weil sie eben so klingen, wie andere Titel, die er bereits gehört hat.

Wie gesagt sollte es sich aber vorerst nur um einen Prototyp des erwähnten Programms handeln. Grund hierfür war, dass dies meine erste Implementierung einer Künstlichen Intelligenz werden sollte und ich vorerst zwiegespalten war, ob das Erlernen des Musikgeschmacks, durch ein Computerprogramm überhaupt möglich ist.

Bis kurze Zeit vor dem Projekt hatte ich bereits sehr viel Erfahrung in Sachen Softwareentwicklung gesammelt und entwickelte etwa zeitgleich meine zweite App. Der Bereich Künstliche Intelligenz und dessen untergeordneter Bereich „Machine Learning" waren mir bis dato aber komplett entgangen. Gerade deshalb hatte ich umso mehr Interesse, mich in diesen Bereich der Informatik einzuarbeiten.

Vorbereitung und Erste Entwürfe

Künstliche Intelligenz

Für den Start in das Thema Künstliche Intelligenz setzte ich mich mit verschiedenen Machine-Learning-Verfahren wie LR (Linear-Regression), KNN (K-Nearest-Neighbor), K-means-Clustering, SVM (Support-Vector-Machine) und auch ein wenig mit neuronalen Netzen auseinander.
Machine Learning ist ein Teilgebiet von KI und ermöglicht es Computersystemen auf Basis von Mustern in Datenbeständen, selbstständig Probleme zu lösen.
Für die genannten Algorithmen implementierte ich nun jeweils ein kleines Beispielprogramm in der Programmiersprache „Python".
Python ist nicht nur meine Erstwahl in Sachen Programmiersprachen, sondern auch besonders gut für Machine Learning geeignet, da sehr viele der Algorithmen in Form von Bibliotheken bereits für Python entwickelt worden sind. Noch dazu ist die Programmiersprache aufgrund ihrer Syntax komfortabler und lesbarer als andere vergleichbare Sprachen.
Mithilfe der Python Bibliothek „Scikit-learn"[1] konnte ich nun die jeweiligen Algorithmen implementieren und auf Beispieldaten, welche online verfügbar sind, anwenden.
Beispielsweise habe ich so für das Erlernen der SVM Methode ein Testprogramm entwickeln können, welches auf Basis von Daten über Patienten eine Erkrankung an Brustkrebs feststellen kann.
SVM ist ein Algorithmus des „Supervised Learning" (auf Deutsch „überwachtes Lernen"), was im Zusammenhang bedeutet, dass der Algorithmus auf Basis gesammelter Daten trainiert wird. Das Training findet zunächst mit den Patientendaten samt ihrer Diagnose statt, später soll der Algorithmus dann selbstständig aus neuen Patientendaten eine Erkrankung feststellen können.
Im Bereich Machine Learning gibt es neben dem Supervised Learning auch noch den Bereich des „Unsupervised Learnings", bei welchem der Algorithmus eigenständig Schlüsse aus den Daten zieht, also ohne sich an bereits gesammelten Musterdaten zu orientieren.

Für die Implementierung der künstlichen Intelligenz zur Ermittlung des Musikgeschmacks wären grundsätzlich mehrere Algorithmen infrage gekommen, ich aber entschied mich, auch hier den SVM Algorithmus zu verwenden, weshalb ich diesen später auch noch im Detail erklären werde.

Kommen wir zunächst zu den ersten Ideen der Implementierung der restlichen Bestandteile des Programms:

Spotify als Datenquelle

Der Audio-Streaming-Dienst Spotify stellt Entwicklern eine vielschichtige API („Application Programming Interface") zur Verfügung. Dies ist eine Schnittstelle einer Anwendung, über welche externe Programme mit dieser kommunizieren können. Anhand dieser Schnittstelle können Programmierer Daten ihres Spotify-Kontos, wie etwa Playlisten, deren Songs und

[1]URL: https://scikit-learn.org/stable/index.html [Besucht am 06.07.2021]

dazugehörige Informationen einsehen und in eigene Anwendungen integrieren. Auch Veränderungen von Playlisten, Auslesen von Informationen wie Titel oder Künstler eines Songs, sowie viele andere Funktionen sind mithilfe der Spotify API möglich.
Die API sollte Teil des Programms werden, um bereits gesammelte Nutzerdaten der KI zur Verfügung zu stellen und später auf Basis des Erlernten unter anderem Playlisten zu erstellen, welche dem Musikgeschmack genau entsprechen.

Von Songs auf Spotify zu Daten für die KI

Um die KI Songs erlernen zu lassen, welche dem Nutzer gefallen, wurden Attribute (Parameter) zur Charakterisierung der Lieder festgelegt. Diese Parameter sollten im Idealfall auch Informationen über Instrumente, Rhythmen oder Gesamtklang des Liedes enthalten, um der KI zu ermöglichen, verschieden klingende Lieder einem Musikgeschmack zuzuordnen.

Meine erste Idee hierfür war, die Audiodateien selbst in ihre Bestandteile zu zerlegen, um sie für die Benutzung durch den Algorithmus zuzuschneiden. Ein Ansatz hierfür ist, alle Frequenzen der Lieder einzeln aus den Dateien zu extrahieren, doch leider ist es trotz des großen Umfangs der Spotify API nicht möglich, die Audiodateien der einzelnen Songs selbst zu erhalten, sondern nur Informationen über diese.
Stattdessen bietet Spotify jedoch eine andere Möglichkeit, welche ich mir zunutze machen konnte: Man kann über die Schnittstelle eine ganze Reihe an sogenannten „Audio Features" einsehen, welche nahezu für jeden Song verfügbar sind und jeweils den Charakter des Songs beschreiben. Hierzu gehören unter anderem:

- BPM = beats per minute (Liedtempo in Schlägen pro Minute)
- danceability = "Tanzbarkeit"
- instrumentalness = Anteil der Passagen ohne Gesang
- valence = musikalische Positivität (fröhlicher - / trauriger Klang)

Eine Liste aller zur Verfügung stehenden Audio Features, sowie deren Beschreibung, ist in Material 1 im Anhang zu finden.
Diese Audio Features (bis auf einige Ausnahmen, wie zum Beispiel die „analysis_url") sagen bereits sehr viel über die musikalische Gestalt von Songs aus und sind über die API einfach zu ermitteln.
Audiodateien könnten natürlich sehr viel mehr über den Typ eines Songs aussagen, wären aber auch mit einem deutlich größeren Aufwand, sowie mit sehr viel mehr Schwierigkeiten in der Implementierung verbunden. Da das Programm aber vorerst nur ein Prototyp sein sollte, entschied ich mich dafür, für das Erste mit den Audio-Features zu arbeiten.

Zunächst führte ich noch einige Analysen meiner Playlisten im Vergleich zu anderen öffentlichen Playlisten durch, um die Audio-Feature-Daten auf ihre Aussagekraft zu testen. In Material 2 ist zur Veranschaulichung die entsprechende Datenmenge für eine meiner Lieblings-Playlisten zu sehen. Der durchschnittliche Song ist unter anderem sehr rhythmisch bzw. in regelmäßigem Tempo (zu erkennen an hoher „danceability"), dem Genre „Rap" zuzuordnen (zu erkennen an niedriger „instrumentalness"), eher düster als fröhlich gestimmt (zu sehen an „valence") und in einem schnellen Tempo von durchschnittlich circa 124 BPM,

welches, nicht aus Zufall, auch etwa dem Tempo entspricht, mit dem ich für gewöhnlich eigene Beats produziere. Songdaten in solch einer Form eignen sich also ganz gut als Trainingsdaten für einen ersten Prototyp.

Implementierung

Programmstruktur

Zunächst einen Blick auf die Verzeichnisstruktur des Prototyp-Programms, in welcher alle benötigten Dateien zu finden sind:

```
soundmatcher/
├─ other/
│  ├─ information.txt
├─ profiles/
│  ├─ 1/
│  │  ├─ info.txt
│  │  ├─ raw_data.txt
│  │  ├─ training_data.txt
├─ settings/
│  ├─ settings.json
├─ current_access_token.txt
├─ main.py
├─ spotify.py
├─ spotify_credentials.py
├─ spotify_song_data_fields.py
```

Bei den Dateien mit der Endung „.py” handelt es sich um ausführbare Python-Dateien. Besonders wichtig sind hierbei die Dateien „main.py” und „spotify.py”, welche alle wichtigen Funktionen und den Großteil des Programmcodes enthalten.
„spotify_credentials.py” beinhaltet Zugangsdaten zu meinem privaten Spotify Konto, die benötigt werden, um mich mit der Spotify API authentifizieren zu können.
„current_access_token.txt” enthält einen weiteren Schlüssel, welcher ebenfalls zur Authentifizierung benötigt wird sowie hierfür automatisch regelmäßig durch das Programm aktualisiert wird. Dieser Schlüssel muss bei jeder Anfrage an die Spotify-Schnittstelle mitgesendet werden.
„spotify_song_data_fields.py” ist nur eine kleine Datei, in jener eine Liste der bereits genannten Audio Features ausgelagert wird, um im Projekt stets Überblick zu gewährleisten.
Das untergeordnete Verzeichnis „settings” enthält eine Datei, in der Einstellungen des Programms zwischen Laufzeiten abgespeichert werden.
“other” ist ein weiteres Verzeichnis, welches jedoch nur für Notizen gebraucht wird und nicht mit dem Programm interagiert.
In „profiles” werden erstellte Profile in weiteren untergeordneten Verzeichnissen abgespeichert, die jeweils zwei Dateien zum Speichern von Daten, die durch den Machine-Learning-Algorithmus berechnet werden, enthalten. In den einzelnen Profilverzeichnissen werden die berechneten Daten abgespeichert, die den Musikgeschmack später beschreiben.

Bei der Programmierung habe ich vorzugsweise einen funktionalen, anstelle eines objektorientierten Programmierstils, genutzt. Das bedeutet, dass der Programmiercode größtenteils auf dem Konzept von Funktionen und dessen Aufrufen, anstatt auf Klassen und Objekten, basiert.

Kommen wir nun zu den besonders wichtigen Dateien „main.py” und „spotify.py”, welche die wichtigsten Komponenten des Projekts sind.
Die „main.py” Datei ist der Startpunkt des Programms und somit die einzige Datei im Verzeichnis, die direkt vom Benutzer (also mir) ausgeführt wird. Die „spotify.py” Datei ist Teil von der „main.py” Datei und implementiert die Dateien: „spotify_credentials.py” und „spotify_song_data_fields.py”.

spotify.py

Diese Datei enthält alle Funktionen, welche die Spotify API abfragen oder für eine Kommunikation mit einem Spotify-Konto notwendig sind. Diese Funktionen liefern unter anderem Informationen über angegebene Playlisten, wie zum Beispiel welche Songs enthalten sind oder fügen neue Songs zu bereits vorhandenen Playlisten hinzu.
Hierfür wird jedes Mal eine Anfrage an die API gestellt, wofür Zugangsdaten benötigt werden. Diese werden ebenfalls von der Datei, mithilfe von anderen Dateien im Verzeichnis, wie „spotify_credentials.py” und „current_access_token.txt”, verwaltet. Noch dazu enthält die Datei Klassen, von denen Objekte instanziiert werden können, die einen Song und dessen Daten repräsentieren und später für den Machine-Learning-Algorithmus benötigt werden.

main.py

Diese Datei ist bildlich gesprochen das Herz des ganzen Programms und somit auch der KI. Hier wird der Machine-Learning-Algorithmus (Support-Vector-Machine) auf die Songdaten, die mithilfe von „spotify.py” geladen werden, angewendet. Damit der Algorithmus die erhaltenen Songdaten verstehen kann und sie effizient, also ohne unnötige Berechnungen, verarbeiten kann, werden diese zugeschnitten und komprimiert.
Des Weiteren können trainierte Daten abgespeichert werden. Was es mit diesen Daten auf sich hat, wird im folgenden Kapitel noch genau erklärt. Es existiert eine Art Verwaltung, welche das Laden und Speichern von verschiedenen gelernten Daten, sozusagen Musikgeschmäckern, ermöglicht.
Ebenso kann in „main.py” auf Basis der erlernten Daten eine Vorhersage getroffen werden, ob ein bestimmter Song, oder gleich mehrere, dem Nutzer gefallen oder nicht gefallen würden.
Nicht zu vergessen existiert eine Benutzeroberfläche in Form eines „Command-Line-Interface”. Hierbei handelt es sich um eine Benutzerschnittstelle für das Programm, welche anders als herkömmlichen Benutzerflächen, im Fachjargon „GUIs”, nur in Form von Text über die Kommandozeile zu bedienen ist. Sie ermöglicht dem Benutzer direkt über bestimmte Kommandos mit den implementierten Funktionen der Datei, und somit auch mit der KI, zu interagieren. (Mehr zum CLI später bei „Anwendung in der Praxis”)

Der Algorithmus „Support Vector Machine”

Kommen wir nun zu dem Algorithmus, welcher im Programm als künstliche Intelligenz agiert. Vorneweg möchte ich jedoch noch erwähnen, dass ich weder im Bereich Machine Learning, noch was die Support-Vector-Machine betrifft, ein Experte bin. Im Folgenden werde ich lediglich mein eigenes Verständnis des Algorithmus erläutern, welches sich aber auch durch entsprechende Ergebnisse als wirksam erwiesen hat.
Wie bereits erwähnt handelt es sich bei SVM um einen Algorithmus des Supervised Learning, was bedeutet, dass er mithilfe von bereits klassifizierten Lerndaten trainiert wird. In unserem Programm bedeutet das, dass der Algorithmus zum Lernen mit Songs gefüttert wird, welche jeweils bereits mit einem der Werte „gut” oder „schlecht” versehen sind. Hier kommt nun der SVM-Algorithmus ins Spiel, welcher als Klassifikator dient, d.h neue Songs, bei welchen noch unbekannt ist, ob sie von dem Nutzer als „gut” oder „schlecht” empfunden werden, möglichst korrekt zuordnet.
Mit den beiden Begriffen sollen natürlich nicht die Songs an sich in irgendeiner Weise bewertet werden. Es geht lediglich darum, die Übereinstimmung mit dem Musikgeschmack entweder als gut oder schlecht zu kennzeichnen.

Nehmen Sie für die folgende Erläuterung der Funktionsweise des Algorithmus das Material 3 zur Hand.
Hier ist zur Demonstration, in einer stark vereinfachten und eindimensionalen Form, eine Vorstufe des SVM Klassifikators zu sehen. Hierbei ist der Einfachheit halber nur die BPM Messung, also das Tempo von Beispieldaten zu sehen, wofür dementsprechend nur eine Dimension benötigt wird. Da der Algorithmus tatsächlich mit 16 verschiedenen Messwerten arbeitet, werden in Wirklichkeit 16 oder sogar mehr Dimensionen berechnet, dazu aber später mehr.
Die grünen Messungen in der Abbildung sollen Songs symbolisieren, welche im Vorhinein als „gut”, d.h. dem Musikgeschmack entsprechend, zugeordnet wurden, die roten wurden als „schlecht” zugeordnet.
Es lässt sich ein Muster erkennen, nämlich dass die Werte über ca. 110 BPM zu den guten Songs gehören, die darunter zu den schlechten. Deshalb wurde nachträglich der blaue Wert hinzugefügt. Dieser symbolisiert den Klassifikator.
Wenn die Daten genauso aussehen, lässt sich nun bereits mit sehr hoher Wahrscheinlichkeit bestimmen, ob ein anderer Song als gut oder schlecht gilt, da zur Überprüfung nur geschaut werden muss, ob der BPM Wert über oder unter 110 BPM liegt.
Nun wurde der Klassifikator bewusst genau zwischen den beiden nächstliegenden Werten verschiedener Klasse, den sogenannten „support vectors”, welche auch den Namen des Algorithmus begründen, platziert, um somit eine möglichst genaue Klassifizierung zu ermöglichen. Der Abstand des Klassifikators und den nächstliegenden Werten wird auch als „margin” bezeichnet.
Sollte einer der Stützvektoren (support vectors) extrem ausfallen und somit den Klassifikator verschlechtern oder sollten sich sogar Gruppierungen von Punkten überlappen, so kann wie in Material 4 dargestellt, eine sogenannte „soft-margin” hinzugefügt werden (gekennzeichnet durch gestrichelte Linien). Die Werte, welche sich in der soft-margin befinden, werden dann vom Klassifikator ignoriert, in der Hoffnung das Ergebnis zu verbessern. Wenn sich Gruppen von Punkten überlappen, können somit die Gruppierungen wieder linear trennbar gemacht werden.

Nun stellt sich natürlich die Frage, wie das Ganze mit mehr als nur einem Messwert funktioniert. Bei mehreren Messwerten wird das Modell um genauso viele Dimensionen, wie Messwerte vorhanden sind, erweitert. Während im Falle einer einzigen Dimension der Klassifikator ein Punkt ist, so ist er bei zwei Dimensionen eine Gerade, bei drei eine Ebene und bei noch höheren Dimensionen eine sogenannte „Hyperebene". In Material 5 ist zur Veranschaulichung ein weiteres Beispiel visualisiert, bei dem zusätzlich zu den BPM Messwerten noch Messwerte der sogenannten „instrumentalness" ergänzt sind. Nun ist der Klassifikator eine blaue Gerade, sozusagen eine Trennlinie.
Wieder werden Werte zwischen den gestrichelten Linien, also innerhalb der soft-margin ignoriert. Der Raum wird auch wieder in zwei Teile geteilt, welche die „guten"- von den „schlechten" Songs trennen.
Es kann auch dazu kommen, dass selbst mit einer größeren soft-margin keine lineare Trennbarkeit erreicht werden kann. Etwa dann, wenn die Werte komplett durcheinander zu liegen scheinen und nicht gruppiert werden können. In diesem Fall wird der sogenannte Kernel-Trick angewendet: Es wird eine weitere Dimension hinzugefügt, dessen Werte durch eine Kernelfunktion berechnet werden. Durch die Ausweitung der Dimensionen erhofft man sich, lineare Trennbarkeit zu erreichen.
In Material 6 ist beispielsweise wieder BPM als einziger Messwert angegeben, doch diesmal sind die Werte nicht linear trennbar. Um trotzdem einen Klassifikator erstellen zu können, wird, wie in Material 7 demonstriert, eine weitere Dimension hinzugefügt, welche durch Anwendung einer Kernelfunktion, nämlich dem Quadrieren der BPM Werte, lineare Trennbarkeit und somit Klassifikation ermöglicht.
Bis hierhin sollte der theoretische Teil der KI verstanden werden, um die Funktionsweise nachvollziehen zu können. Letzten Endes muss der Algorithmus nicht von Hand implementiert werden, sondern lediglich mit entsprechenden Parametern angewendet werden, weil entsprechende Bibliotheken bereits vorhanden sind.
Wichtig zu verstehen ist jedoch, dass Berechnungen in Wahrheit deutlich komplexer sind, als in den dargestellten Visualisierungen. Das Prinzip ist zwar das gleiche, doch statt den dargestellten ein oder zwei Dimensionen, sind es in der Praxis dieses Programms mindestens 16, da so viele Audio-Features benutzt werden und vermutlich noch einige Dimensionen dazu gerechnet werden.
Zusätzlich wird die KI in der Praxis nicht nur mit 10 bis 20, sondern idealerweise mit einigen tausend Songdaten trainiert. Dementsprechend wird auch einiges an Rechenleistung benötigt, um das Training in einer vernünftigen Zeit absolvieren zu können.

Anwendung in der Praxis

Nun ist der erste Prototyp bereit für die Anwendung beziehungsweise einige Praxistests. Werfen wir hierfür zunächst ein Blick auf die Bedienung des Programms:

Bedienung des Prototyps

Sobald das Programm über die „main.py" Datei gestartet wird, erscheint ein Konsolenfenster, über welches per „Command-Line-Interface" mit dem Programm interagiert werden kann. Wie in Material 8 dargestellt, werden beim Start mehrere Befehle angezeigt, welche zur Verfügung stehen, um mit dem Programm zu interagieren.
Da es sich, wie bereits erwähnt, für das Erste nur um einen Prototyp handelt, sind vor allem Befehle vorhanden, welche zum Testen des Programms benötigt werden. Eine benutzerfreundliche Oberfläche ist noch nicht vorhanden.
Folgende Befehle stehen zur Verfügung:

1. Mit dem Befehl „a" kann eine Playlist geladen und mit einem trainierten Musikgeschmack abgeglichen werden. Hierfür wird jeder Song der Playlist samt einer Vorhersage, ob der Song auf Basis des Musikgeschmacks als gut oder als schlecht empfunden wird, angezeigt. Je nach Konfiguration des Programms können die Titel, welche als gut vorhergesagt werden, direkt zu einer eigenen Playlist auf Spotify hinzugefügt werden.

2. Mit dem Befehl „s" kann das aktuell ausgewählte Profil bearbeitet werden. Wird das Profil verändert, so kann man hier Playlisten angeben, auf Basis welcher dann der Algorithmus trainiert und ein Klassifikator erstellt werden soll.

3. Der Befehl „d" dient dazu, das aktuell ausgewählte Profil zu ändern. Hierfür muss ein Pfad zu dem neuen Profilverzeichnis angegeben werden.

4. Mit dem Befehl „f" kann die Einstellung konfiguriert werden, ob bei Vorhersage von Songs in einer Playlist durch den Befehl „a" die Titel, welche als gut vorhergesagt wurden, automatisch zu einer bestimmten „eigenen Playlist" hinzugefügt werden sollen.

5. Der Befehl „g" dient dazu, die eben genannte, sogenannte „Output Playlist" zu ändern. Dafür muss die Spotify-URI der neuen Playlist angegeben werden, welche über das Spotify Programm selbst verfügbar ist.

6. Mit „h" können alle Songs aus der „Output Playlist" entfernt werden.

7. Mit „j" werden unter Angabe einer Playlist-URI, die entsprechenden Audio Features der gesamten Playlist angezeigt (Siehe Material 2).

Ein Testdurchlauf

Nun ist es Zeit für einen Testdurchlauf. Hierfür erstelle ich zunächst ein neues Profil mit dem Namen „test".

Um bestmögliche Ergebnisse zu erzielen, sammle ich so viele Daten in Form von Playlisten wie möglich. Benötigt werden sowohl möglichst viele Playlisten mit guten, als auch möglichst viele mit schlechten Songs. Ob die Playlisten als gut oder schlecht empfunden werden, muss bekannt sein, da die Zuordnung der Trainingsdaten für den Algorithmus notwendig ist. Als „gute Playlisten" wähle ich meine 26 zuletzt erstellten Playlisten aus meiner Mediathek. Dafür kopiere ich alle Spotify-URIs der Playlisten aus Spotify und speichere diese ab. Die Playlisten sollten meinen Musikgeschmack recht gut repräsentieren, da ich zu eigenen Playlisten natürlich nur Titel hinzufüge, die mir wirklich gefallen.
Bei den schlechten Playlisten ist die Auswahl etwas schwerer zu treffen. Gesucht sind natürlich Songs, die meinem Musikgeschmack nicht entsprechen. Theoretisch könnte sich also fast jeder Song, welcher sich nicht in einer meiner privaten Playlisten befindet, mehr oder weniger gut als Trainingsdatenmenge für den Algorithmus eignen. Selbstverständlich gibt es auch Songs, die zwar meinem Musikgeschmack entsprechen, aber nicht in meinen Playlisten vorhanden sind. Dementsprechend sind Songs aus Genres, die weniger meinem Musikgeschmack entsprechen, möglicherweise ganz gut geeignet.
Außerdem gehe ich auf Basis meines Wissens über Support-Vector-Machine davon aus, dass besonders Songs, die auf dem großen Spektrum von musikalischen Typen und Genres eher meinem Geschmack entsprechen, aber immer noch meiner Meinung nach für den Algorithmus als „schlechte" Songs gelten sollen, in Kombination mit den guten Songs, besonders gute Ergebnisse erzielen können. Ich vermute, dass gerade durch sehr ähnliche, aber anders zugeordnete Daten, ein besonders präziser Klassifikator erstellt werden kann, während „schlechte" Songs mit komplett anderen Daten den Klassifikator eventuell nicht genau genug bestimmen.
Ein weiterer Punkt ist, dass die für den Algorithmus schlechten Songs aus möglichst vielen verschiedenen Genres kommen sollten. Schließlich soll der eigene Musikgeschmack aus möglichst allen Genres hervorgehoben werden und es soll nicht nur zwischen dem Musikgeschmack und beispielsweise einem einzigen anderen Genre unterschieden werden. Die genannten Punkte können die Resultate der KI sicherlich positiv beeinflussen, sind aber nicht zwingend notwendig.
Für den Test entscheide ich mich also dafür, einige öffentliche Playlisten verschiedener Genres auszuwählen, welche von Spotify unter dem jeweiligen Genre vorgeschlagen werden. Insgesamt suche ich 23 Playlisten heraus, welche unter verschiedenen Genres zu finden sind. Auch einige Rap Songs sind dabei, welche jedoch nicht genau meinem Musikgeschmack entsprechen, und sich somit vermutlich gut für den Algorithmus eignen.
Um als nächstes die KI trainieren zu können, muss das aktuelle Profil mit dem Befehl „d" zum neu erstellten, mit dem Namen „test" gewechselt werden (Siehe Material 9).
Es werden zwei Fehlermeldungen angezeigt, welche darauf verweisen, dass bestimmte Dateien im Profilverzeichnis fehlen, da das Verzeichnis eben noch keine abgespeicherten Daten der KI enthält. Da wir im Folgenden erst die KI trainieren werden, können wir diese Meldungen ignorieren.
Nach dem Ausführen des Befehls „s" mit dem die KI konfiguriert wird, wird man durch ein Menü geleitet (Siehe Material 10). Man bekommt nun die Auswahl, ob man bereits vorhandenen Songdaten aus dem Profil laden möchte oder neue erstellen möchte. In unserem Fall existieren noch gar keine Daten im Profil, weshalb ich mit „new" ein neues Datenmodell erstelle. Infolgedessen werde ich aufgefordert, die „guten" sowie die „schlechten" Songs, in Form von Playlisten bzw. deren Spotify-URIs, anzugeben. Bei mehreren Playlisten, wie in unserem Fall, müssen die URIs durch Kommas separiert werden.

Danach werden alle Songs in den Playlisten geladen. Es wird nun angezeigt, dass insgesamt 653 gute und 1721 schlechte Songs geladen wurden (Siehe Material 11).
Anschließend werden für alle Songs die entsprechenden Audio-Feature-Daten geladen. Da nicht für alle Songs die Daten verfügbar und vollständig sind, werden bei fehlenden Daten entsprechende Meldungen in der Konsole angezeigt.
In Material 12 ist dann zu sehen, dass aufgrund fehlender Daten insgesamt 47 Songs entfernt wurden. Vergleichsweise sind das nicht sehr viele Songs, weshalb das kein großes Problem darstellt.
Des Weiteren werden im Hintergrund die Daten auch noch stark komprimiert, damit möglichst viele Ressourcen gespart werden können.
Das Laden der über 2000 Songs, mit jeweils 16 Messwerten, sowie die Verarbeitung dieser Daten, dauert auf meiner recht leistungsstarken Hardware insgesamt ca. 14 Minuten und 20 Sekunden.
Im nächsten Schritt wird nach dem Laden oder Erstellen von Trainingsdaten gefragt. Wieder erstellen wir neue Daten, weil noch keine vorhanden sind. Die erstellten Daten werden nun gespeichert und der Klassifikator wird erstellt.
Danach wird die Genauigkeit des Klassifikators angegeben. Hierfür wird im Hintergrund ein kleiner Teil der bereits zugeordneten Songdaten, welche fürs Training gesammelt wurden, vorbehalten, um den Klassifikator mit diesen abzugleichen und ihn auf diese Weise auf seine Genauigkeit zu testen. Hierbei ergibt sich eine Genauigkeit von ungefähr 91%. Ein recht guter Wert.
Anschließend steht dem Benutzer die Möglichkeit offen, einige Testdaten ausgeben zu lassen. Ich entscheide mich jedoch dagegen, weil wir die KI im Folgenden auf anderem Weg testen werden.
Jetzt ist es soweit: Die KI ist auf den Musikgeschmack trainiert und einsatzbereit für einige Vorhersagen. Fangen wir erst einmal mit einer leichten Aufgabe an: Die KI soll bei jedem Song in einer Schlager-Playlist vorhersagen, ob dieser mir wohl gefällt oder nicht. Da mein Musikgeschmack nun wirklich nicht mit Schlager übereinstimmt, sollte hierbei kein einziger Song von der KI als „gut” vorhergesagt werden. Hierfür wähle ich die erstbeste Playlist auf Spotify aus, welche unter dem Genre Schlager vorgeschlagen wird. Eine Playlist namens „Schlager Klassiker”.
Mit dem Befehl „a” des Programms wird eine Vorhersage von den Songs in der Playlist gestartet. Der Prozess dauert nur wenige Sekunden. Das Resultat ist in Material 13 zu sehen. Alle Songs der Playlist werden als „schlecht”, also als nicht übereinstimmend mit dem Musikgeschmack, vorhergesagt. Die KI hat die Songs also alle richtig vorhergesagt.
Eine zweite und deutlich schwerere Aufgabe soll nun zeigen, wie gut die KI zwischen guten und schlechten Songs unterscheiden kann. Hierfür erstelle ich eine Playlist, in welche ich abwechselnd gute und schlechte Songs hinzufüge. Alle Songs sind natürlich nicht in einer der Playlisten enthalten, welche zum Trainieren genutzt wurden, sondern komplett neu für die KI.
Die schlechten Songs entnehme ich hierbei aus einer Playlist namens „Disco Hi-Life”. Ich wähle diese Playlist, weil dadurch der Algorithmus möglicherweise mehr herausgefordert wird, da solch ein Genre vermutlich eher mit meinem Musikgeschmack zu verwechseln ist, als beispielsweise das Schlagergenre. Es soll also schwerer für die KI sein, diese Songs richtig vorherzusagen.
Bei den guten Songs wähle ich solche, die mir gefallen, ich aber nicht in Playlisten gespeichert hatte, da ich viele von diesen ja bereits für das Training benutzt hatte.

Die Ergebnisse sind in Material 14 zu sehen: Bis auf einen Song („Go” von Huncho Jack) wurden alle Songs richtig vorhergesagt.
Dass dieser eine Titel falsch vorhergesagt wurde, kann daran liegen, dass er ziemlich ruhig im Vergleich zu meinem und dementsprechend auch zu dem erzeugtem Musikgeschmack ist und deshalb wohl nicht genug den guten Songs der Trainingsdaten entspricht. Dennoch hätte ich persönlich diesen Song meinem Musikgeschmack zugeordnet.

Auswertung der Ergebnisse & Ausblick für die Zukunft

Nachdem ich nun längere Zeit das Programm getestet habe, kann ich sagen, dass immer wieder ähnliche Ergebnisse, wie in den Tests, zustande gekommen sind.
Statt wie früher die von Spotify empfohlenen Playlisten im Hip-Hop / Rap Genre selbst zu durchwühlen, übernahm nun das Programm diese Aufgabe. Hierfür wurden die Songs, welche die KI als meinem Musikgeschmack entsprechend vorhersagte, in eigene, eben gefilterte Playlisten kopiert, bevor ich diese zu Ohren bekam.
Leider gab es ab und zu auch ein paar wenige Songs, welche die KI für mich auswählte, die mir nicht so wirklich gut gefielen. Der Grund für diese wenigen Ausnahmen war jedoch recht einfach: Die der KI zugrundeliegenden Daten waren eben nicht die echten Audiodateien der Songs, sondern nur die 16 Audio Features, welche die Songs vergleichsweise einfach nicht so detailliert beschreiben konnten.
Dennoch erzielte die KI im Großen und Ganzen ziemlich gute Ergebnisse und automatisiert mir den Prozess der Liedersuche bis heute noch fast komplett.
Der Prototyp ist vielleicht nicht perfekt, aber es ist eben noch ein Prototyp. Wichtig für mich ist vor allem, dass er deutlich gezeigt hat, dass die zugrunde liegende Idee umsetzbar ist und somit das eigentliche Ziel, einer KI zur Bestimmung des Musikgeschmacks, erreichbar ist.
Ich bin fest davon überzeugt, dass eine weiter ausgebaute KI, welche mit detaillierteren Daten der Songs arbeitet, das Potenzial hat, Musikgeschmäcker von Menschen deutlich besser, bis hinzu perfekt, zu bestimmen.
Auch bei dem SVM Algorithmus gibt es sicher noch viele Möglichkeiten, diesen zu optimieren. Es gibt noch viele Stellschrauben, welche durch Optimierung den Algorithmus verbessern könnten.
Ebenso könnte dieser Algorithmus auch durch andere Machine Learning Algorithmen ersetzt werden, welche in diesem Szenario eventuell bessere Ergebnisse erzielen könnten.
Auch denkbar wäre, Algorithmen zu verwenden, welche aus ganz anderen Bereichen von Machine Learning stammen. So könnten voraussichtlich Algorithmen aus dem Bereich des Unsupervised- oder Reinforcement Learnings, den Musikgeschmack ohne das Angeben von guten und schlechten Songs im Vorhinein, erlernen, da diese sehr viel eigenständiger lernen können.
Somit könnte das Programm auch deutlich komfortabler gemacht werden und müsste theoretisch nur noch im Hintergrund laufen. Playlisten mit den entsprechenden Songs könnten dann komplett automatisiert erstellt werden.
Für mich steht jedenfalls fest, dass ich über den Prototyp hinaus noch weiter an dem Projekt arbeiten werde und hoffentlich bald auch eine erste öffentliche Version zur Verfügung stellen kann.

Fazit

Im Großen und Ganzen würde ich das Projekt auf jeden Fall als erfolgreich bezeichnen. Mir ist nicht nur der Einstieg in das Thema KI und Machine Learning gelungen, sondern ich habe auch die Erkenntnis gewonnen, dass Computerprogramme sehr wohl in der Lage sein können, den Musikgeschmack von Menschen zu erlernen. Dementsprechend kann ich nun auch meine Ausgangsfrage, ob eine KI intelligent genug sein kann, einen Musikgeschmack zu erlernen, mit einem klaren „Ja" beantworten.
Auf Basis dessen sowie auf Basis des bereits entwickelten Prototyps, kann ich nun die Projektidee weiter umsetzen.
Darüber hinaus habe ich auch gelernt, wie mächtig KIs sein können.
Vor der Einarbeitung in das Thema waren für mich Computer nicht viel mehr als bloße Rechner. Eine Intelligenz in dieser Form damit entwickeln zu können, erschien mir vorerst überhaupt nicht möglich zu sein.
Noch dazu sehe ich für KI auch große Chancen in der Zukunft. Prozessoren werden immer schneller, Speichermedien bekommen immer mehr Speicherplatz. In den Praxistests des Programms, bei denen ich Songdaten von weit über 2000 Songs mit der KI trainierte, dauerte dies auf meinem System nur ca. 14 min. Vor nicht allzu langer Zeit, hätte derselbe Prozess aufgrund wesentlich schlechterer Hardware noch deutlich länger gedauert. In Zukunft wird er dementsprechend wahrscheinlich auch deutlich schneller sein und schon heute gibt es erste Quantencomputer, die durch modernste Berechnungsverfahren in vermutlich nur wenigen Sekunden den Prozess absolvieren könnten. Aus schnelleren Berechnungen in kürzerer Zeit und größeren Speichermedien werden somit auch intelligentere KIs folgen, da diese somit mehr und in kürzerer Zeit lernen können.
Es ist also nur noch eine Frage der Zeit, wann KIs nicht mehr aus unserem Alltag wegzudenken sind und uns unsere neuen Lieblingssongs auf Spotify vorspielen.

Anhang

Material 1:

Key	Type
`acousticness` A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.	Float
`analysis_url` An HTTP URL to access the full audio analysis of this track. An access token is required to access this data.	String
`danceability` Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.	Float
`duration_ms` The duration of the track in milliseconds.	Integer
`energy` Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.	Float
`id` The Spotify ID for the track.	String
`instrumentalness` Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.	Float
`key` The key the track is in. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on.	Integer
`liveness` Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.	Float
`loudness` The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.	Float
`mode` Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.	Integer
`speechiness` Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.	Float
`tempo` The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.	Float
`time_signature` An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).	Integer
`track_href` A link to the Web API endpoint providing full details of the track.	String
`type` The object type: "audio_features"	String
`uri` The Spotify URI for the track.	String
`valence` A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).	Float

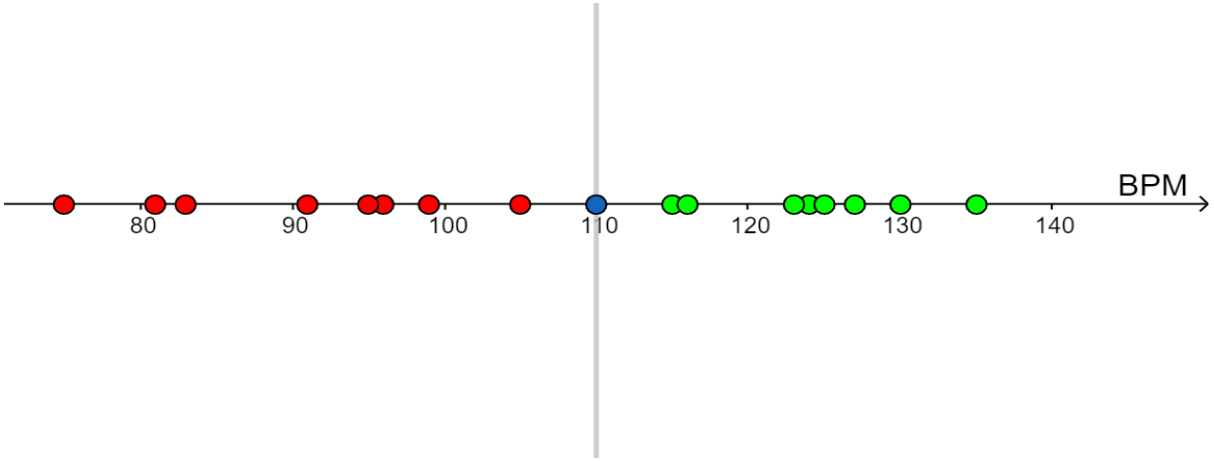
Übersicht der Attribute zur Klassifizierung der Songs auf Spotify, den sogenannten „Audio Features"
(Quelle: https://developer.spotify.com/documentation/web-api/reference/#category-tracks)

Material 2:

```
                    Playlist (2TWWj1mAeWsGG7u6zJ1ebW)
{   'acousticness': 0.155,
    'danceability': 0.827,
    'energy': 0.58,
    'instrumentalness': 0.015,
    'key': 5.037,
    'key_confidence': 0.423,
    'liveness': 0.152,
    'loudness': -7.387,
    'mode': 0.5,
    'mode_confidence': 0.459,
    'speechiness': 0.219,
    'tempo': 124.045,
    'tempo_confidence': 0.554,
    'time_signature': 4.0,
    'time_signature_confidence': 0.961,
    'valence': 0.418}
```

Ausgabe der durchschnittlichen „Audio Features” einer Playlist meines Musikgeschmacks

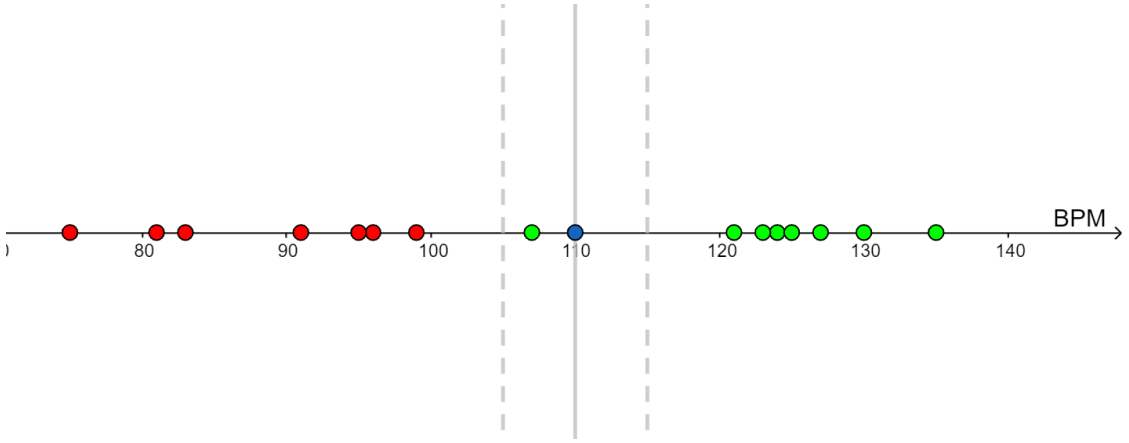
Material 3:



Visualisierung einer stark vereinfachten und eindimensionalen Form des SVM-Klassifikators anhand des Messwertes „BPM”

blau: Klassifikator
grün: „gute”, d.h. dem Musikgeschmack entsprechende Messwerte
rot: schlechte”, d.h. dem Musikgeschmack nicht entsprechende Messwerte

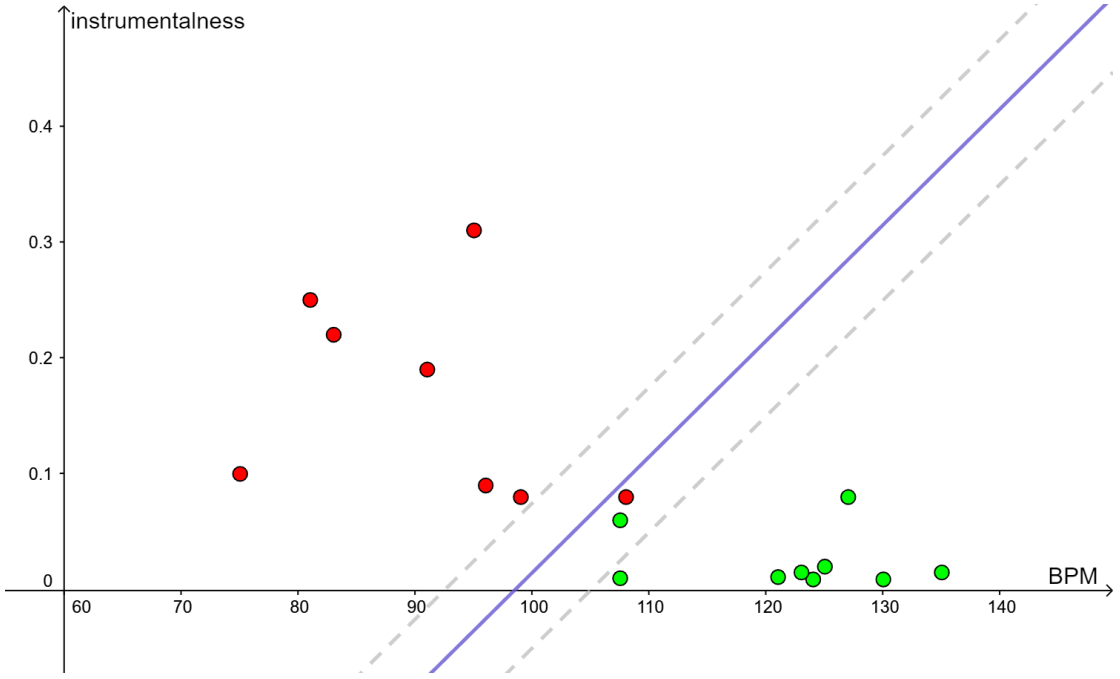
Material 4:



Visualisierung der „soft-margin" des SVM-Klassifikators

blau: Klassifikator
grün: „gute", d.h. dem Musikgeschmack entsprechende Messwerte
rot: schlechte", d.h. dem Musikgeschmack nicht entsprechende Messwerte

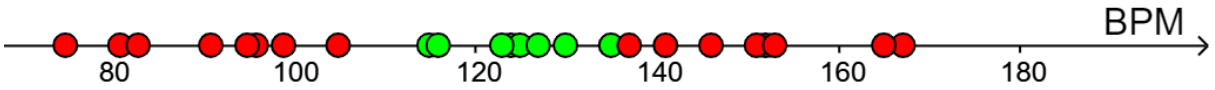
Material 5:



Visualisierung einer stark vereinfachten, zweidimensionalen Form des SVM-Klassifikators anhand der Messwertes „BPM" und „instrumentalness"

blau: Klassifikator
grün: „gute", d.h. dem Musikgeschmack entsprechende Messwerte
rot: schlechte", d.h. dem Musikgeschmack nicht entsprechende Messwerte

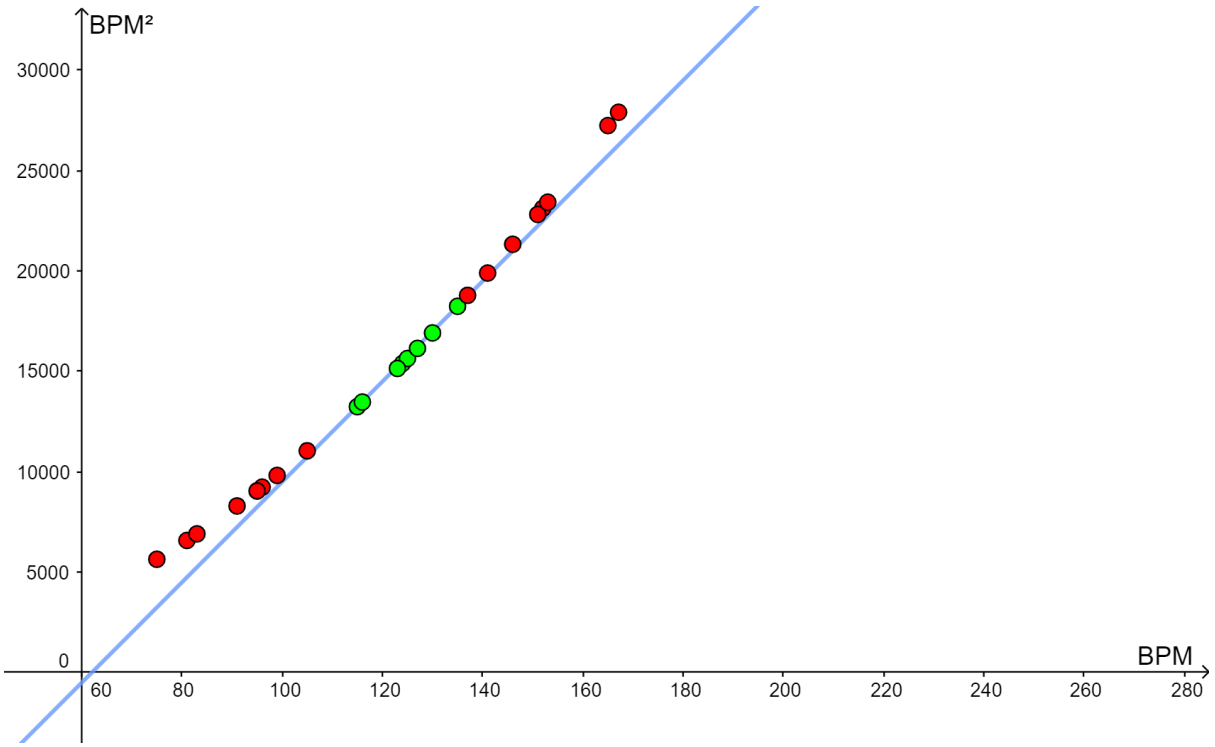
Material 6:



Visualisierung einer nicht linear trennbaren Datenmenge des Messwertes „BPM”

grün: „gute”, d.h. dem Musikgeschmack entsprechende Messwerte
rot: schlechte”, d.h. dem Musikgeschmack nicht entsprechende Messwerte

Material 7:



Visualisierung der linearen Trennbarkeit von einer Datenmenge durch Anwendung eines „Kernel-Tricks” (Quadrieren des BPM-Wertes)

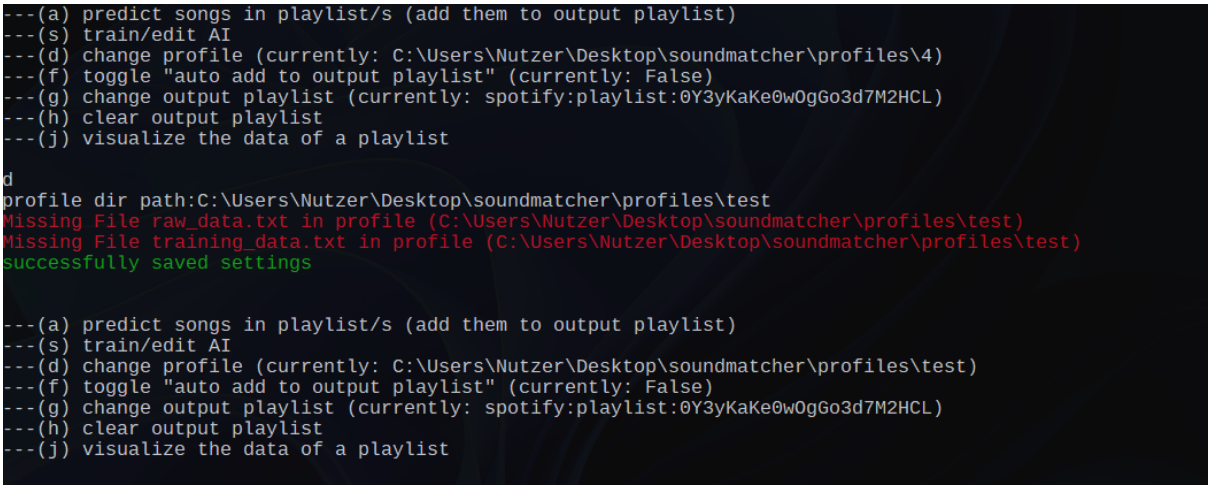
blau: Klassifikator
grün: „gute”, d.h. dem Musikgeschmack entsprechende Messwerte
rot: schlechte”, d.h. dem Musikgeschmack nicht entsprechende Messwerte

Material 8:

```
---(a) predict songs in playlist/s (add them to output playlist)
---(s) train/edit AI
---(d) change profile (currently: C:\Users\Nutzer\Desktop\soundmatcher\profiles\4)
---(f) toggle "auto add to output playlist" (currently: True)
---(g) change output playlist (currently: spotify:playlist:0Y3yKaKe0wOgGo3d7M2HCL)
---(h) clear output playlist
---(j) visualize the data of a playlist
```

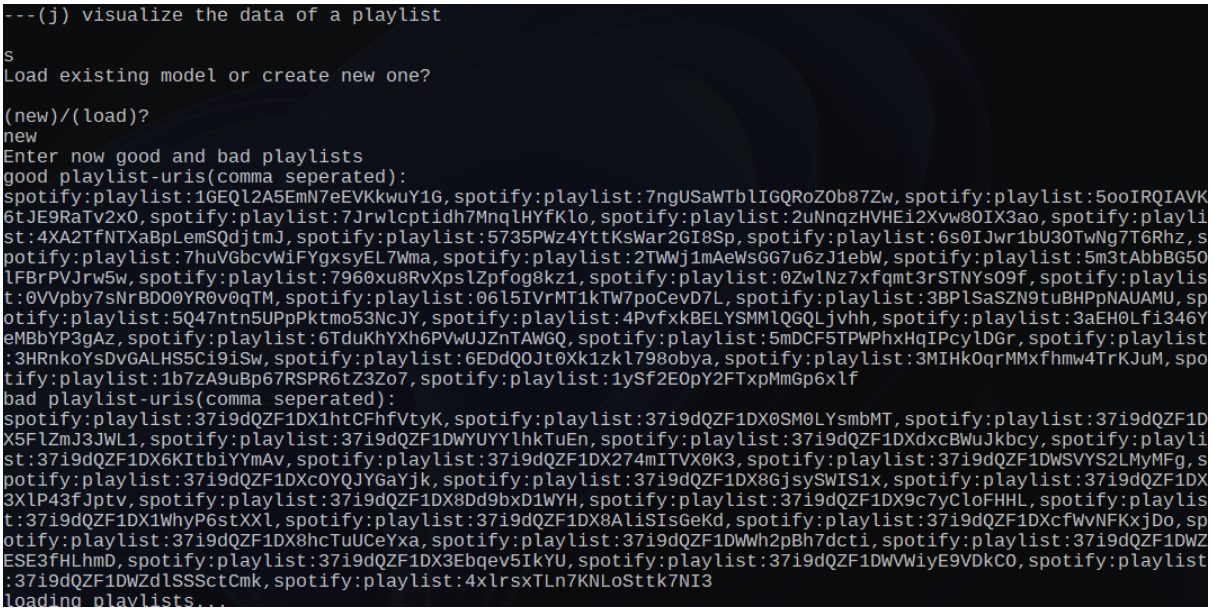
CLI („Command-Line-Interface”) der Anwendung

Material 9:



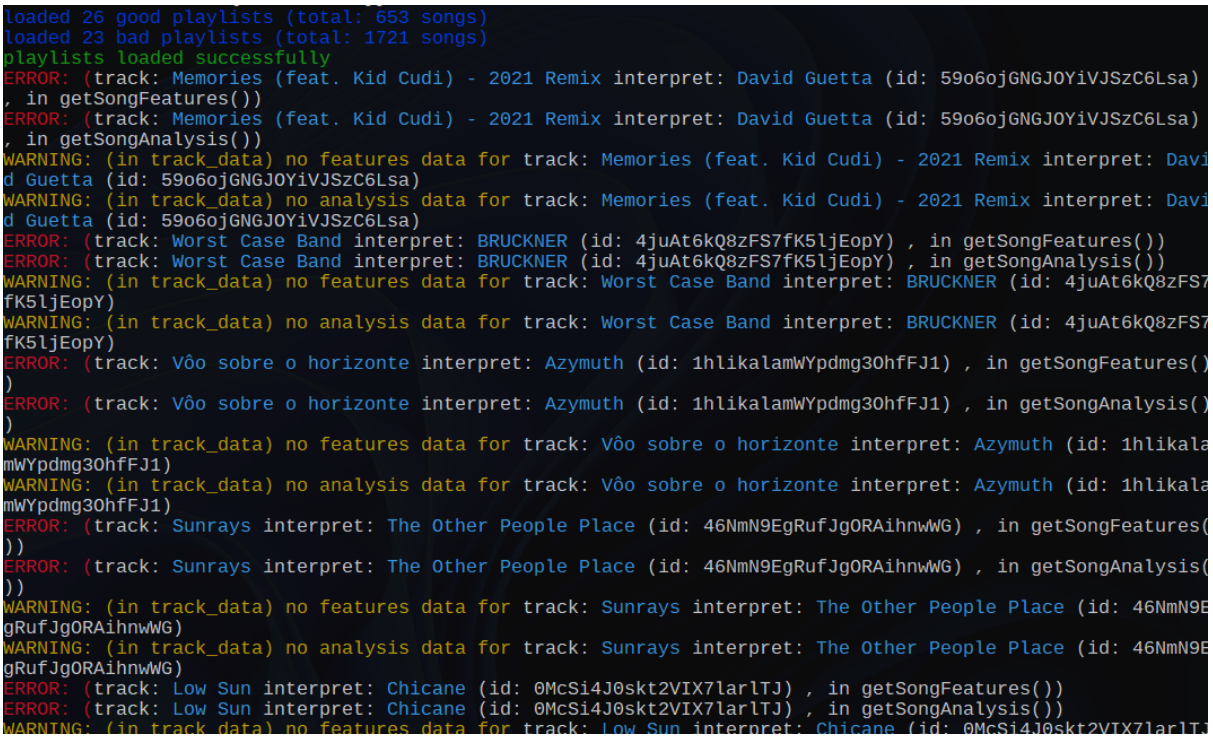
CLI Ausgabe bei Profilwechsel (Befehl „d”)

Material 10:



CLI Ausgabe beim Erstellen eines neuen Profils auf Basis von „Spotify-Playlist-URIs" (Befehl „s")

Material 11:



CLI Ausgabe während des Ladens der Spotify-Songs

Material 12:

```
WARNING: (in track_data) no analysis data for track: Off My Mind interpret: reezy (id: 669SFq0ZPiEsXFzHs9tC
ar)
Removed 47 Songs because of uncomplete datasets!
successfully loaded settings
successfully saved raw data
Load existing training data or create new?
(new)/(load)?
new
successfully loaded settings
successfully saved training data
Creating new CLF...
Successfully created new CLF
current accuracy: 0.9098712446351931
Log a data test?

(yes)/(no)?
no


---(a) predict songs in playlist/s (add them to output playlist)
---(s) train/edit AI
---(d) change profile (currently: C:\Users\Nutzer\Desktop\soundmatcher\profiles\test)
---(f) toggle "auto add to output playlist" (currently: False)
---(g) change output playlist (currently: spotify:playlist:0Y3yKaKe0wOgGo3d7M2HCL)
---(h) clear output playlist
---(j) visualize the data of a playlist
```

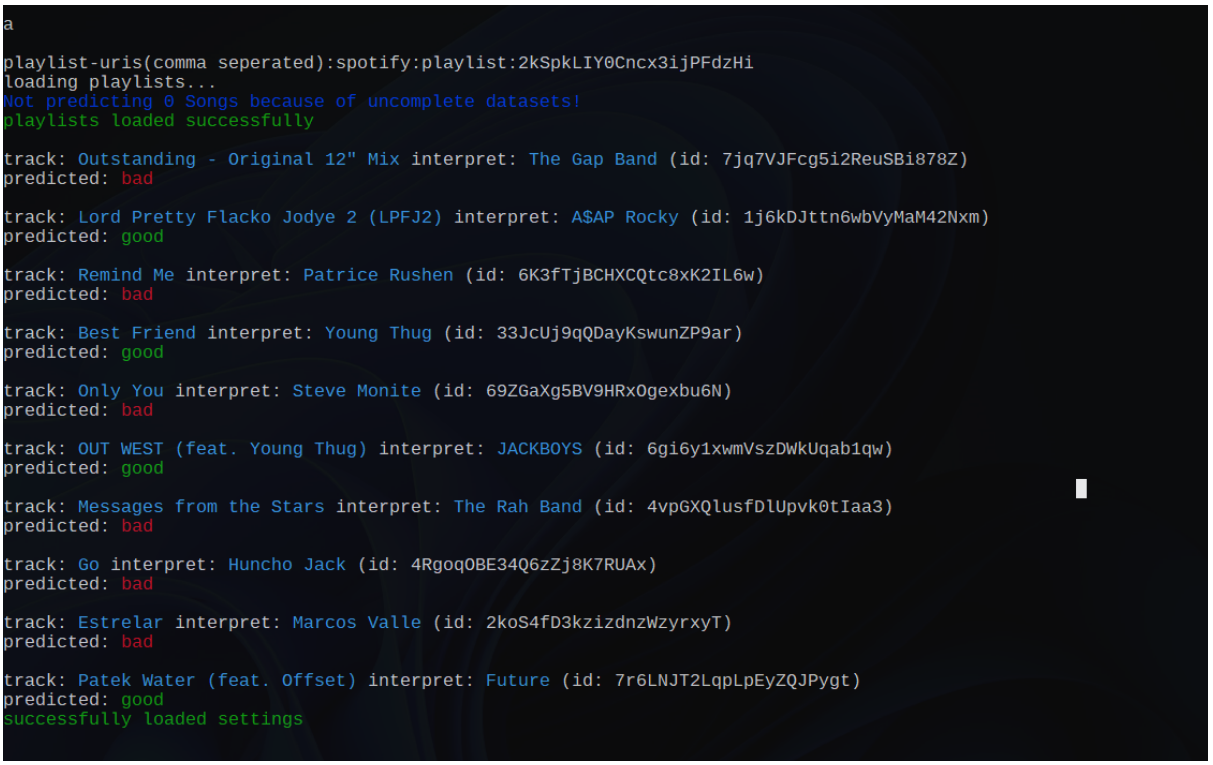
ƒCLI Ausgabe während der Erstellung des SVM-Klassifikators

Material 13:

```
a

playlist-uris(comma seperated):spotify:playlist:37i9dQZF1DX6uHioFvkN7A
loading playlists...
Not predicting 0 Songs because of uncomplete datasets!
playlists loaded successfully

track: Ein bisschen Spaß muss sein interpret: Roberto Blanco (id: 5bYpoFFTImibvRIf54KdKq)
predicted: bad

track: Griechischer Wein interpret: Udo Jürgens (id: 452WSSzuklpg0w4RKTlkvn)
predicted: bad

track: Lotosblume interpret: Die Flippers (id: 6Q4jqkegYVAewgSxOXFXRv)
predicted: bad

track: Der Junge mit der Mundharmonika - '94er Version interpret: Bernd Clüver (id: 2w95s63oOJKZPKRcKAfOrN)
predicted: bad

track: Johnny Blue interpret: Lena Valaitis (id: 5e6KfjLDK12b19zGnHaDf4)
predicted: bad

track: Michaela interpret: Bata Illic (id: 2yvm5QDnxQ6ST3Qp4spW90)
predicted: bad

track: An Der Nordseeküste interpret: Klaus & Klaus (id: 7lg9reolractKDt8McON0K)
predicted: bad

track: Ich bin verliebt in die Liebe interpret: Chris Roberts (id: 4lb2leZpHhBMza7TInQbQo)
predicted: bad

track: Lotosblume interpret: Die Flippers (id: 0EC1wzovpK3rX36SGDZtNj)
predicted: bad

track: Joana interpret: Roland Kaiser (id: 2P1J5TuisANfrOyFPtS7RO)
predicted: bad

track: Siebzehn Jahr, blondes Haar interpret: Udo Jürgens (id: 6RTEzkbTQCivNPX5ShoJVv)
predicted: bad

track: Die Gefühle haben Schweigepflicht interpret: Andrea Berg (id: 5vFTxd3r00Ecx2MwOW6NiV)
predicted: bad

track: Er steht im Tor interpret: Wencke Myhre (id: 6Xpd2qTFOFdJMp8A2Y3QwX)
predicted: bad

track: Tür an Tür mit Alice interpret: Howard Carpendale (id: 5xhHDc9Il8aUrAu4ijixDg)
predicted: bad
```

CLI Ausgabe bei Vorhersage der Übereinstimmung von Songs einer Schlager-Playlist mit meinem Musikgeschmack (Befehl „a")

Material 14:



CLI Ausgabe bei Vorhersage der Übereinstimmung von Songs einer Playlist namens „Disco Hi-Life” mit meinem Musikgeschmack (Befehl „a”)

Quellen

Buchquellen:

- Raschka, Sebastian, Mirjalili, Vahid (2021): *Machine Learning mit Python und Keras*, 3. Auflage, Frechen, mitp Verlag GmbH & Co.KG

- Weigend, Michael, (2019): *Python 3, Lernen und professionell anwenden. Das umfassende Praxisbuch*, 8. Auflage, Frechen, mitp Verlag GmbH & Co.KG

Internetquellen:

- Wikipedia (2021): *Nächste-Nachbarn-Klassifikation*, Online im Internet: URL: https://de.wikipedia.org/wiki/N%C3%A4chste-Nachbarn-Klassifikation [Besucht am 02.07.2021]

- Luber ,Stefan, Litzel, Nico (2016): *Was ist Machine Learning?*, Online im Internet: URL: https://www.bigdata-insider.de/was-ist-machine-learning-a-592092/ [Besucht am 02.07.2021]

- Dokumentation der Python Bibliothek, *scikit-learn,* Online im Internet: URL: https://scikit-learn.org/stable/index.html [Besucht am 06.07.2021]

- Machine Learning Repository, *Breast Cancer Wisconsin (Diagnostic) Data Set,* Online im Internet: URL: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) [Besucht am 06.07.2021]

- Wuttke, Laurenz, *Machine Learning: Definition, Algorithmen, Methoden und Beispiele*, Online im Internet: URL: https://datasolut.com/was-ist-machine-learning/ [Besucht am 02.07.2021]

- Spotify API Dokumentation, Online im Internet: URL: https://developer.spotify.com/documentation/web-api/reference/#category-tracks [Besucht am 07.07.2021]

- Youtube, StatQuest with Josh Starmer, *Support Vector Machines Part 1 (of 3): Main Ideas!!!,* Online im Internet: URL: https://www.youtube.com/watch?v=efR1C6CvhmE [Besucht am 10.07.2021]

- Youtube, Tech with Tim, *Python Machine Learning Tutorials,* Online im Internet: URL: https://www.youtube.com/playlist?list=PLzMcBGfZo4-mP7qA9cagf68V06sko5otr [Besucht am 10.07.2021]

- Wikipedia (2021): *Support Vector Machine*, Online im Internet: URL: https://de.wikipedia.org/wiki/Support_Vector_Machine [Besucht am 10.07.2021]

- Spotify Playlist, *Schlager Klassiker*, auf Spotify: URL: https://open.spotify.com/playlist/37i9dQZF1DX6uHioFvkN7A?si=5221cc86a43b4c51 [Besucht am 16.07.2021]

- Spotify Playlist, *Disco Hi-Life*, auf Spotify: URL: https://open.spotify.com/playlist/37i9dQZF1DXbS8bPVXXR2B?si=a6ade26093824b6f [Besucht am 16.07.2021]

- Youtube, Tagesschau, *Superrechner: Erster Quantencomputer in Deutschland vorgestellt,* Online im Internet: URL: https://www.youtube.com/watch?v=PSshTCmTrDo [Besucht am 17.07.2021]

Tool für Visualisierungen:

- GeoGebra, Online im Internet: URL: https://www.geogebra.org/ [Besucht am 15.07.2021]

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Frankfurt am Main, den 06.11.2021
Felician Schwarz