Machine Learning zum Cocktailmixen

Florian Grassl

Fachsemester: 5

Betreuer: Moritz Fuchsloch, Kim Glumann
Institut für Strahlwerkzeuge (IFSW)
Universität Stuttgart

Stuttgart, 10. März 2025

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Seit der Veröffentlichung von GPT-4 im November 2022 hat das öffentliche Interesse an Künstlicher Intelligenz (KI) und Maschinellem Lernen (ML) drastisch zugenommen. Neben klassischen Feldern wie Bildverarbeitung oder Sprachverstehen stoßen „unorthodoxe“ Anwendungsbereiche zunehmend auf Interesse. Ein solches Beispiel ist das **automatisierte Mischen von Cocktails**: Hier treffen unterschiedliche Geschmackspräferenzen (z. B. „süß“ vs. „sauer“) auf eine komplexe Vielfalt an Zutaten. Machine Learning kann helfen, diesen mehrdimensionalen Raum zu strukturieren und personalisierte Vorschläge zu generieren.

Cocktails sind nicht nur ein kulinarischer Genuss, sondern repräsentieren auch ein nichtlineares, hochdimensionales Geschmacksproblem. Unterschiedliche Zutaten bringen unterschiedliche Intensitäten in den Dimensionen „süß“, „sauer“, „bitter“, „fruchtig“, „würzig“ usw. mit sich. Gleichzeitig hat jeder Mensch subjektive Vorlieben, die nicht immer einfach zu quantifizieren sind. Die zentrale Fragestellung dieser Arbeit lautet: „Wie kann man mit Methoden des maschinellen Lernens individuelle Geschmacksprofile erfassen und daraus geeignete Mischverhältnisse für Cocktails ableiten?“

1.2 Stand der Technik

Im Bereich des maschinellen Lernens dominieren mittlerweile diverse Modellklassen für unterschiedliche Einsatzzwecke. **Transformer-Modelle** wie in [VSPea17] haben die natürliche Sprachverarbeitung revolutioniert. Zur Generierung realistisch wirkender Daten, z. B. synthetischer Bilder, kommen **Generative Adversarial Networks** (GANs) [GPAMea14] zum Einsatz. Ein **Autoencoder** [GBC16] ist hingegen ein unüberwachtes Verfahren, das bei der Rekonstruktion latenter Datenmuster punkten kann – insbesondere dann, wenn die Datensätze nicht allzu groß sind. Zudem könnte ein

Reinforcement-Learning-Ansatz [SB18] langfristig angewendet werden, um Feedback (z. B. Nutzerbewertungen) direkt in das System zurückzuführen und es iterativ zu verbessern.

Kapitel 2

Hauptteil

2.1 Methodik

2.1.1 Datenvorverarbeitung

Ein wichtiger Baustein ist die Vorbereitung der Daten. Dazu wurde eine JSON-Datei angelegt, welche sämtliche Zutaten mit ihren zugehörigen Geschmacksprofilen enthält. Beispielhaft liegen pro Zutat fünf Werte vor: „süß“, „sauer“, „bitter“, „fruchtig“, „würzig“ sowie ein Boolean `alcoholic`, der kennzeichnet, ob eine Zutat Alkohol enthält oder nicht. Die Dimensionen sind dabei beliebig erweiterbar – bei Bedarf könnte man „salzig“, „umami“ oder „kühlend“ ergänzen.

Um die Daten zu normalisieren, wird häufig der `StandardScaler` (`scikit-learn`) verwendet, der jeden Wert auf Mittelwert 0 und Standardabweichung 1 skaliert. Diese Normalisierung verhindert, dass einzelne Geschmacksdimensionen das Training dominieren, nur weil sie größere Zahlenwerte aufweisen.

2.1.2 Fragebogen und Nutzerprofil

Der Nutzer füllt im Programm einen Fragebogen aus, wobei für jede „Entweder-Oder“-Frage ein kleiner Geschmacksvektor hinterlegt ist. Beispielsweise bedeutet „Süß“ $=(1,0,0,0,0)$, „Sauer“ $=(0,1,0,0,0)$. Nach mehreren Fragen wird das individuelle Profil $\mathbf{p}_{\text{user}}$ als Mittelwert dieser Teilvektoren ermittelt:

$$\mathbf{p}_{\text{user}} \;=\; \frac{1}{N}\sum_{i=1}^{N}\mathbf{q}_i.$$

Während manche Nutzer stark „süß“ und „fruchtig“ tendieren, können andere eher „herb“ und „bitter“ favorisieren. Parallel wird abgefragt, ob der Drink „alkoholisch“ sein soll und wie viele Zutaten k maximal erlaubt sind.

2.1.3 Autoencoder zur Profilrekonstruktion

Ein **Autoencoder** eignet sich hervorragend, um latente Strukturen zu erkennen und eventuell „rauschartige“ Komponenten zu glätten. Er besteht aus:

- **Encoder**: Komprimiert das Nutzerprofil $\mathbf{p}_{\text{user}}$ auf einen Latent-Space (z. B. drei Dimensionen).
- **Decoder**: Rekonstruiert aus dem Latent-Space das Profil $\mathbf{p}_{\text{rekon}}$.

Das Ziel im Training ist es, den *Mean Squared Error* (MSE) zwischen $\mathbf{p}_{\text{rekon}}$ und dem Originalprofil $\mathbf{p}_{\text{user}}$ zu minimieren.

Beispielcode für einen Autoencoder in Python

Listing 2.1: Einfacher Autoencoder-Aufbau mit Keras

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

def build_and_train_autoencoder(ingredient_profiles, latent_dim=3,
                                epochs=500, batch_size=4):
    input_dim = ingredient_profiles.shape[1]  # 5 Dimensionen?

    # Encoder
    input_layer = Input(shape=(input_dim,))
    encoded = Dense(latent_dim, activation='relu')(input_layer)

    # Decoder
    decoded = Dense(input_dim, activation='sigmoid')(encoded)

    # Autoencoder zusammensetzen
    autoencoder = Model(inputs=input_layer, outputs=decoded)
    autoencoder.compile(optimizer='adam', loss='mse')

    # Training
    autoencoder.fit(
        ingredient_profiles,
        ingredient_profiles,
        epochs=epochs,
```

```
        batch_size=batch_size,
        verbose=1
    )
    return autoencoder
```

2.1.4 Least-Squares-Optimierung

Nach der Rekonstruktion entsteht ein „idealisiertes“ Profil $\mathbf{p}_{\text{rekon}}$. Um daraus die tatsächlichen Mischverhältnisse der Zutaten abzuleiten, verwendet man eine **Least-Squares**-Optimierung:

$$\min_{\mathbf{w}} \|\mathbf{A}\,\mathbf{w} - \mathbf{p}_{\text{rekon}}\|^2,$$

wobei $\mathbf{A} \in \mathbb{R}^{5\times M}$ alle verfügbaren Zutatenprofile enthält und $\mathbf{w} \in \mathbb{R}^M$ die Gewichte (Volumenanteile) sind. Negative Lösungen werden auf 0 gesetzt, und nur die Top-k Gewichte bleiben übrig.

Codebeispiel für die Mischung

Listing 2.2: Berechnung der Mischprofile mit Least Squares

```
import numpy as np

def create_mix_profile(reconstructed_profile, ingredient_profiles,
                       ingredient_names, k, total_volume=200.0):
    # Matrix A = Transpose der Zutatenprofile
    A = ingredient_profiles.T
    b = reconstructed_profile

    # Löst unbeschränktes Least-Squares
    w, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

    # Kein negatives Volumen
    w = np.maximum(w, 0)

    # Normieren, sum(w)=1
    sum_w = np.sum(w)
    if sum_w == 0:
        w = np.ones_like(w) / len(w)
    else:
        w /= sum_w
```

```
    # Nur Top-k Gewichte behalten
    if k < len(w):
        sorted_idx = np.argsort(-w)
        top_k_idx = sorted_idx[:k]
        mask = np.zeros_like(w, dtype=bool)
        mask[top_k_idx] = True
        w[~mask] = 0
        sum_w2 = np.sum(w)
        if sum_w2 > 0:
            w /= sum_w2
        else:
            w[mask] = 1.0 / k

    # ml-Umskalierung (z.B. 200 ml)
    volumes = w * total_volume
    mixture = {}
    for name, vol in zip(ingredient_names, volumes):
        if vol > 1e-6:
            mixture[name] = round(float(vol), 2)
    return mixture
```

2.1.5 Aufteilung in alkoholisch/nicht-alkoholisch

Wenn ein **alkoholischer** Cocktail gewünscht wird, wird beispielsweise 30 % des Gesamtvolumens für alkoholische Zutaten reserviert und 70 % für nicht-alkoholische. Dazu teilt man die Zutatenprofile in zwei Gruppen (alkoholisch vs. nicht-alkoholisch) und führt pro Gruppe eine Least-Squares-Berechnung durch. Am Ende werden beide Gruppen kombiniert.

2.2 Konkreter Programmablauf und Beispiele

Das System lädt zunächst die JSON-Datei, in der `ingredients` und `questions` definiert sind. Dann beantwortet der Nutzer die Fragen. Ein exemplarisches JSON-Fragment (inkl. Fragebogen) sieht so aus:

Listing 2.3: Exemplarische JSON mit Fragen und Zutaten

```
{
  "ingredients": {
    "Zuckersirup": {
```

```
        "taste": [0.9, 0.0, 0.0, 0.1, 0.0],
        "alcoholic": false
      },
      "Zitronensaft": {
        "taste": [0.0, 0.9, 0.1, 0.0, 0.0],
        "alcoholic": false
      },
      "Gin": {
        "taste": [0.2, 0.1, 0.4, 0.0, 0.3],
        "alcoholic": true
      }
    },
    "questions": [
      {
        "question": "Möchten Sie es eher süß oder sauer?",
        "option1_text": "Süß",
        "option1_taste": [1, 0, 0, 0, 0],
        "option2_text": "Sauer",
        "option2_taste": [0, 1, 0, 0, 0]
      },
      {
        "question": "Bevorzugen Sie eine alkoholische
          Variante?",
        "option1_text": "Ja",
        "option1_taste": [0, 0, 0, 0, 0],  // separately
          handled
        "option2_text": "Nein",
        "option2_taste": [0, 0, 0, 0, 0]
      }
    ]
}
```

Nach dem Durchlauf des Fragebogens steht das `user_profile`. Dann fragt das Programm in der Konsole, ob der Cocktail „alkoholisch“ oder „nicht alkoholisch“ sein soll, und wie viele Zutaten man maximal (k) verwenden möchte. Mithilfe des `autoencoder`-Modells wird $\mathbf{p}_{\text{rekon}}$ erstellt und anschließend per `create_mix_profile` das finale Mischungsverhältnis bestimmt.

2.2.1 Rating-Funktion

Der Nutzer kann abschließend eine Bewertung (z.B. auf einer Skala von 1 bis 5) abgeben. Diese wird in einer separaten JSON-Datei gespeichert, was perspektivisch für *Supervised Learning* oder *Reinforcement Learning* genutzt werden kann. Ein Beispiel:

Listing 2.4: Speichern einer Nutzerbewertung in einer JSON-Datei

```
import os, json
from datetime import datetime

def save_rating(user_profile, cocktail_mix, rating):
    ratings_file = "cocktail_ratings.json"

    # Falls Datei nicht existiert, leere Struktur anlegen
    if not os.path.exists(ratings_file):
        with open(ratings_file, "w", encoding="utf-8") as f:
            json.dump({"ratings": []}, f, ensure_ascii=False, indent=2)

    with open(ratings_file, "r", encoding="utf-8") as f:
        data = json.load(f)

    new_entry = {
        "timestamp": datetime.now().isoformat(timespec="seconds"),
        "user_profile": list(map(float, user_profile)),
        "cocktail_mix": cocktail_mix,
        "rating": rating
    }

    data["ratings"].append(new_entry)

    with open(ratings_file, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)

    print("Deine Bewertung wurde erfolgreich gespeichert!")
```

Auf diese Weise können langfristig detaillierte Nutzerfeedbacks gesammelt und analysiert werden.

2.3 Ergebnisse und Diskussion

Erste Tests zeigen, dass **Autoencoder + Least-Squares-Optimierung** in der Lage sind, funktionierende Rezeptvorschläge zu generieren. Nutzer, die „süß“ und „fruchtig“ angaben, bekamen häufiger Säfte und Sirup empfohlen, wohingegen „bitter“-orientierte Personen Kräuterliköre oder Gin-dominierte Drinks erhielten. Die „alkoholisch / nicht-alkoholisch“-Trennung erlaubte eine saubere Kontrolle über den Anteil alkoholischer Zutaten.

Dennoch treten einige **Limitationen** auf:

- *Datenqualität*: Fehlen zu viele Zutaten in der JSON-Datei, entstehen Schätzungen, die möglicherweise nicht akkurat oder „leer“ wirken.
- *Extremprofile*: Personen mit sehr „extremen“ Antworten (z.B. „nur süß, 0% bitter“) können Rezepte erhalten, die zu einseitig schmecken.
- *Feedback-Loop*: Das System ist (noch) nicht dynamisch „lernend“; es könnte aber dank gespeicherter Ratings in Zukunft iterativ verfeinert werden.

2.4 Zusammenfassung und Ausblick

Diese Arbeit verdeutlicht, wie Machine-Learning-Techniken auf den Anwendungsfall „Cocktailmixen“ übertragen werden können. Ein **Autoencoder** rekonstruiert das Geschmacksprofil, **Least-Squares** bestimmt die Gewichtsanteile der Zutaten, und Nutzerfeedback kann in JSON-Dateien gespeichert werden, um das System perspektivisch anzupassen. Zu den möglichen Erweiterungen gehören:

- **Weitere Geschmacksdimensionen**: z.B. „salzig“, „umami“, „kühlend“.
- **Reinforcement Learning**: Echtzeit-Anpassungen basierend auf Nutzerbewertungen („Reward“).
- **Datenbasis ausbauen**: Mehr Zutaten, ggf. mit Sensorik-Informationen aus Smart-Dispensern.
- **Gastronomische Nutzung**: Automatisierte Cocktailmaschinen, die direkt auf Nutzervorlieben reagieren.

Obwohl das Modell den menschlichen Geschmackssinn nicht ersetzen wird, zeigt es, dass KI in der Lage ist, hochdimensionale Geschmacksprobleme zu strukturieren und dem Anwender konkrete Rezeptvorschläge zu liefern. Die vorgestellte Lösung ist zudem modular und skalierbar. Das bedeutet, sie lässt sich leicht erweitern, wenn neue Zutaten oder geänderte Geschmacksdimensionen hinzukommen.

Kapitel 3

Literaturverzeichnis

Literaturverzeichnis

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GPAMea14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, and et al. Generative adversarial nets. Proceedings of NIPS, 2014.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[VSPea17] Ashish Vaswani, Noam Shazeer, Niki Parmar, and et al. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

Selbstständigkeitserklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe. Alle verwendeten Hilfsmittel, Quellen und Zitate sind ordnungsgemäß angegeben und kenntlich gemacht.

Stuttgart, 10. März 2025 (Florian Grassl)