O'REILLY®

US-Bestseller
Verständlich durch
farbige Illustrationen

Praxisbuch Large Language Models

Sprache mit KI verarbeiten und generieren

Jay Alammar &
Maarten Grootendorst

Übersetzung von Marcus Fraaß

Vorwort

Large Language Models (LLMs), auch als große Sprachmodelle bezeichnet, verändern die Welt grundlegend. Dank LLMs können Maschinen menschliche Sprache nicht nur besser verstehen, sondern auch selbst generieren. Dadurch eröffnen sich völlig neue Möglichkeiten im Bereich der künstlichen Intelligenz (KI), die bereits zahlreiche Branchen maßgeblich beeinflussen und auch in Zukunft weiter verändern werden.

Dieses Buch bietet Ihnen eine ebenso umfassende wie anschauliche Einführung in die Welt der LLMs – beginnend mit den theoretischen Grundlagen bis hin zu einer Vielzahl praktischer Anwendungen. Dabei lernen Sie wichtige Entwicklungen im Zusammenhang mit LLMs kennen, darunter einige, die dem Deep Learning vorausgingen, wie die Darstellung von Wörtern als Vektoren. Zudem erfahren Sie mehr über die Transformer-Architektur, die maßgeblich zum Erfolg von LLMs beigetragen hat. Sie erhalten außerdem einen Einblick in die Funktionsweise von LLMs und lernen die verschiedenen Architekturen sowie Techniken zum Trainieren und Feintunen kennen. Darüber hinaus gewinnen Sie einen umfassenden Überblick über die vielfältigen Anwendungsbereiche von LLMs, wie beispielsweise die Textklassifikation, das Clustering, das Topic Modeling, Chatbots, Suchmaschinen und vieles mehr.

Mit diesem Buch, das Ihnen durch zahlreiche praktische Anwendungen und Illustrationen einen leicht verständlichen Zugang zur Thematik bietet, möchten wir die ideale Grundlage für alle schaffen, die die aufregende Welt der LLMs erkunden möchten. Ganz gleich, ob Sie gerade erst in die Materie einsteigen oder bereits Experte sind: Tauchen Sie ein in die Welt der LLMs und lernen Sie, wie Sie diese selbst erstellen können.

Ein leicht verständlicher Zugang

Das Hauptziel dieses Buchs ist es, Ihnen einen *leicht verständlichen* Zugang zum Thema LLMs zu verschaffen. Da sich die Fortschritte im Bereich der Language AI – also KI-basierter Sprachanwendungen – in einem geradezu atemberaubenden Tempo vollziehen, kann es herausfordernd sein, mit den neuesten Entwicklungen Schritt zu

halten. Aus diesem Grund liegt der Schwerpunkt dieses Buchs auf den Grundlagen von LLMs, um Ihnen einen unterhaltsamen und einfachen Lernprozess zu ermöglichen.

Um Ihnen das Thema *so nachvollziehbar wie möglich* zu vermitteln, begegnen Ihnen im Laufe des Buchs zahlreiche Illustrationen. Mithilfe dieser Illustrationen sollen Ihnen die wichtigsten Konzepte und Techniken im Zusammenhang mit LLMs auf anschauliche Weise nähergebracht und der Einstieg in dieses spannende und potenziell weltverändernde Gebiet erleichtert werden.[1]

Im gesamten Buch wird strikt zwischen sogenannten Representation-Modellen und generativen Sprachmodellen unterschieden. Bei Representation-Modellen handelt es sich um LLMs, die nicht dazu dienen, Text zu generieren, sondern vielmehr darauf ausgelegt sind, Texte bestmöglich abzubilden. Sie werden häufig für aufgabenspezifische Anwendungsfälle wie die Klassifikation verwendet. Generative Modelle sind hingegen LLMs, die Texte generieren, wie beispielsweise GPT-Modelle. Obwohl einem generative Modelle in der Regel als Erstes in den Sinn kommen, wenn man an LLMs denkt, bieten Representation-Modelle ebenfalls ein breites Spektrum an Anwendungsmöglichkeiten. Im weiteren Verlauf wird das Wort »Large« (groß) in *Large Language Models* relativ vage verwendet. Zum Teil werden sie auch einfach als Sprachmodelle bzw. Language Models bezeichnet, da Abgrenzungen hinsichtlich ihrer Größe oft willkürlich vorgenommen werden und nur bedingt Rückschlüsse auf die Leistungsfähigkeit von Modellen ermöglichen.

An wen sich dieses Buch richtet

In diesem Buch wird vorausgesetzt, dass Sie bereits über Programmiererfahrung in Python verfügen und sich mit den Grundlagen des maschinellen Lernens auskennen. Der Schwerpunkt liegt weniger auf der mathematischen Herleitung von Gleichungen als vielmehr auf der Vermittlung eines guten, intuitiven Verständnisses. Zu diesem Zweck werden in diesem Buch zahlreiche Abbildungen und viele praktische Beispiele angeführt, die die Lerninhalte veranschaulichen und verständlicher machen. Vorkenntnisse bezüglich gängiger Deep-Learning-Frameworks wie PyTorch oder TensorFlow oder im Bereich der Erstellung generativer Modelle sind für dieses Buch nicht erforderlich.

Sollten Sie noch keine Erfahrung im Umgang mit Python haben, empfehlen wir Ihnen als Einstieg die Onlinetutorials von Learn Python (*https://oreil.ly/arcIm*), die die wesentlichen Grundlagen der Programmiersprache vermitteln. Um Ihnen den Lernprozess so einfach wie möglich zu gestalten, steht Ihnen der gesamte Code auf Google Colab (*https://oreil.ly/kSucO*) zur Verfügung – einer Plattform, auf der sich der Code ausführen lässt, ohne dass eine lokale Installation erforderlich ist.

1 J. Alammar. »Machine learning research communication via illustrated and interactive web articles.« *Beyond Static Papers: Rethinking How We Share Scientific Understanding in ML*. ICLR 2021 Workshop (2021).

Aufbau des Buchs

Das Buch ist grundsätzlich in drei Hauptteile gegliedert. Abbildung 1 bietet Ihnen einen Gesamtüberblick über den Inhalt der einzelnen Kapitel. Diese können auch unabhängig voneinander gelesen werden. Daher können Sie die Kapitel, mit deren Inhalt Sie bereits vertraut sind, problemlos überfliegen.

Teil I: Die Funktionsweise von Sprachmodellen verstehen

In Teil I des Buchs wird die Funktionsweise kleiner und großer Sprachmodelle näher beleuchtet. Zunächst wird ein Überblick über das Fachgebiet und gängige Methoden vermittelt (siehe Kapitel 1). Anschließend werden zwei der wichtigsten Bausteine dieser Modelle, Tokenisierung und Embeddings, vorgestellt (siehe Kapitel 2). Abgerundet wird dieser Teil des Buchs durch eine aktualisierte und um weitere Inhalte ergänzte Fassung von Jays bekanntem Beitrag *The Illustrated Transformer* (*https://oreil.ly/UI4lN*), in der detailliert auf die Architektur dieser Modelle eingegangen wird (siehe Kapitel 3). Dabei werden zahlreiche Begriffe eingeführt und grundlegende Konzepte erläutert, die im weiteren Verlauf des Buchs immer wieder von Bedeutung sein werden.

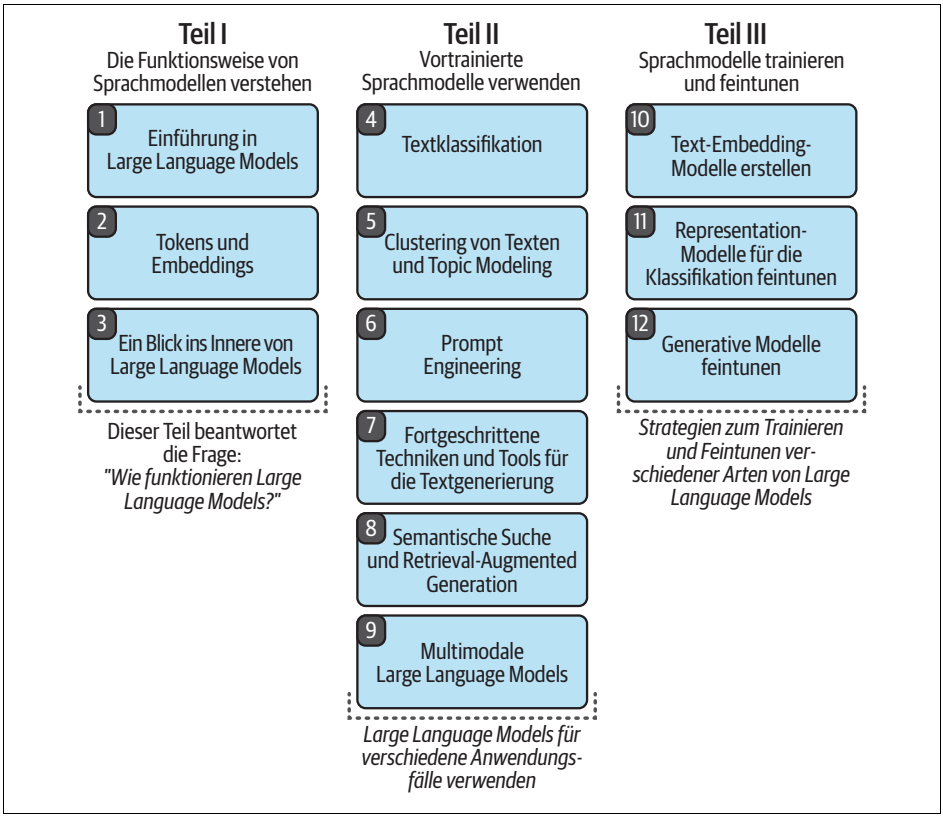

Abbildung 1: Übersicht über die einzelnen Teile und Kapitel des Buchs

Teil II: Vortrainierte Sprachmodelle verwenden

In Teil II werden anhand gängiger Anwendungsfälle verschiedene Einsatzmöglichkeiten von LLMs aufgezeigt. Dabei wird auf vortrainierte Modelle zurückgegriffen und gezeigt, wozu diese bereits ohne weiteres Feintuning imstande sind.

Sie erfahren, wie Sie Sprachmodelle zur überwachten Klassifikation (siehe Kapitel 4), zum Clustering von Texten und zum Topic Modeling (siehe Kapitel 5) verwenden können. Zudem erfahren Sie, wie Sie Texte generieren (siehe die Kapitel 6 und 7) und Embedding-Modelle für die semantische Suche nutzen können (siehe Kapitel 8). Abschließend können Sie sich ein Bild davon machen, wie sich die Möglichkeiten im Bereich der Textgenerierung auf Bilder übertragen lassen (siehe Kapitel 9).

Nachdem Sie sich mit diesen grundlegenden Anwendungsmöglichkeiten von Sprachmodellen vertraut gemacht haben, sind Sie in der Lage, eigenständig Problemstellungen verschiedenster Art zu lösen und zunehmend ausgefeiltere Systeme und Pipelines zu entwickeln.

Teil III: Sprachmodelle trainieren und feintunen

In Teil III des Buchs lernen Sie anspruchsvollere Ansätze kennen und erfahren, wie sich Sprachmodelle trainieren und feintunen lassen. Sie lernen, wie Sie ein Embedding-Modell erstellen und feintunen (siehe Kapitel 10), wie Sie ein BERT-Modell feintunen können, um eine Klassifikation durchzuführen (siehe Kapitel 11), und schließlich auch, wie Ihnen das Feintuning generativer Modelle gelingt (siehe Kapitel 12).

Hardware- und Softwarevoraussetzungen

Generative Modelle auszuführen, erfordert in der Regel eine hohe Rechenleistung und somit einen Computer mit einem leistungsstarken Grafikprozessor (engl. *Graphics Processing Unit*, GPU) bzw. einer leistungsstarken Grafikkarte. Da nicht alle Leserinnen und Leser über einen solchen Computer verfügen, sind alle Beispiele in diesem Buch so konzipiert, dass sie über eine online verfügbare Plattform namens Google Colaboratory (*https://oreil.ly/HQawv*) (Google Colab) ausgeführt werden können. Zum Zeitpunkt des Verfassens dieses Buchs können Sie auf dieser Plattform kostenfrei eine T4 von NVIDIA verwenden, um den Code auszuführen. Diese GPU verfügt über 16 GB Grafikspeicher (engl. *Video Random-Access Memory*, VRAM), was der mindestens erforderlichen Speichergröße für die meisten der im Buch enthaltenen Codebeispiele entspricht.

Nicht für alle Kapitel ist ein Grafikspeicher von mindestens 16 GB erforderlich, da einige Beispiele, wie das Trainieren und Feintunen von Sprachmodellen, rechenintensiver sind als andere, etwa das Prompt Engineering. Die genauen Mindestvoraussetzungen hinsichtlich des Grafikspeichers für die einzelnen Kapitel können Sie dem Repository entnehmen.

Den gesamten Code, die erforderlichen Voraussetzungen und zusätzliche Tutorials finden Sie im Repository dieses Buchs (*https://github.com/HandsOnLLM/Hands-On-Large-Language-Models*). Wenn Sie die Beispiele lokal ausführen wollen, empfehlen wir Ihnen die Verwendung einer Grafikkarte von NVIDIA mit einem Grafikspeicher von mindestens 16 GB. Möchten Sie eine lokale Installation, beispielsweise mit `conda`, durchführen, empfehlen wir Ihnen, Ihre Umgebung wie folgt einzurichten:

```
conda create -n thellmbook python=3.10
conda activate thellmbook
```

Sie können alle erforderlichen Abhängigkeiten installieren, indem Sie das Repository forken oder klonen und dann Folgendes in Ihrer neu erstellten Python-3.10-Umgebung ausführen:

```
pip install -r requirements.txt
```

API-Schlüssel

In den Beispielen verwenden wir sowohl Open-Source- als auch proprietäre Modelle, um Ihnen die jeweiligen Vor- und Nachteile vor Augen zu führen. Um die in diesem Buch vorgestellten proprietären Modelle von OpenAI und Cohere zu verwenden, müssen Sie zunächst ein kostenloses Nutzerkonto erstellen:

OpenAI (https://oreil.ly/M4nAa)
: Klicken Sie auf der Website auf die Schaltfläche *Sign up*, um ein kostenloses Nutzerkonto zu erstellen. Mit diesem können Sie einen API-Schlüssel erzeugen, mit dem Sie später auf GPT-3.5 zugreifen können. Klicken Sie dann auf das Settings-Symbol und anschließend auf *API keys*, um einen geheimen Schlüssel zu erzeugen.

Cohere (https://oreil.ly/T63GA)
: Erstellen Sie zunächst ein kostenloses Nutzerkonto auf der Website. Klicken Sie dann auf die Schaltfläche *API keys*, um einen geheimen Schlüssel zu erzeugen.

Bei beiden Nutzerkonten gibt es allerdings Einschränkungen bezüglich der Anzahl der API-Aufrufe, das heißt, mit diesen kostenlosen API-Schlüsseln können Sie nur eine begrenzte Anzahl von Aufrufen pro Minute durchführen. Wir haben diesen Umstand bei allen Beispielen berücksichtigt und falls erforderlich entsprechende lokal ausführbare Alternativen bereitgestellt.

Zum Verwenden der Open-Source-Modelle müssen Sie kein Nutzerkonto erstellen. Eine Ausnahme bildet das in Kapitel 2 verwendete Llama-2-Modell, das Sie nur mit einem Hugging-Face-Nutzerkonto verwenden können:

Hugging Face (https://oreil.ly/_uV3A)
: Klicken Sie auf der Website von Hugging Face auf *Sign up*, um ein kostenloses Nutzerkonto zu erstellen. Wählen Sie anschließend im Menü *Settings* die Option *Access Tokens* aus und erstellen Sie ein Token, das Sie zum Herunterladen bestimmter LLMs verwenden können.

Anwendungsfälle mit deutschsprachigen Daten

Die in diesem Buch verwendeten Open-Source-Modelle wurden von den Autoren hinsichtlich ihrer Performance bei den vorgestellten englischsprachigen Codebeispielen ausgewählt (neben weiteren Aspekten wie beispielsweise den erforderlichen Rechenressourcen, um eine Ausführbarkeit über Google Colab zu gewährleisten). Wenn Sie die Codebeispiele mit deutschen Prompts durchspielen oder auch eigene Anwendungsfälle mit deutschsprachigen Daten umsetzen wollen, sollten Sie auf andere Modelle ausweichen, die für den jeweiligen Anwendungsfall besser geeignet sind. Die in den Codebeispielen verwendeten Modelle können Sie problemlos austauschen, indem Sie ein anderes Modell angeben. Je nach Aufgabengebiet kommen eine Vielzahl mehr- bzw. deutschsprachiger Modelle infrage, wie derzeit z.B. Qwen3 (*https://hugging face.co/models?search=qwen3*), Gemma 3 (*https://huggingface.co/models?search=gemma3*), XLM-RoBERTa (*https://huggingface.co/models?search=xlm-roberta*) oder auch EuroBERT (*https://huggingface.co/models?search=eurobert*). Die Suche nach einem geeigneten Modell können Sie am einfachsten über den Model Hub von Hugging Face unter *https://huggingface.co/models?language=de&sort=trending* durchführen. Dort können Sie auch weitere Attribute wie z.B. die von Ihnen verfolgte Aufgabe (unter »Tasks«) oder die Größe des Modells (»Parameters« unter »Main«) angeben.

In diesem Buch verwendete Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateiendungen.

`Konstante Zeichenbreite`
: Wird für Programmlistings und für Programmelemente in Textabschnitten wie Namen von Variablen und Funktionen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter verwendet.

`Konstante Zeichenbreite, fett`
: Kennzeichnet Befehle oder anderen Text, den die Nutzerin bzw. der Nutzer wörtlich eingeben sollte.

`Konstante Zeichenbreite, kursiv`
: Kennzeichnet Text, den die Nutzerin bzw. der Nutzer je nach Kontext durch entsprechende Werte ersetzen sollte.

Dieses Symbol steht für einen Tipp oder eine Empfehlung.

KAPITEL 4

Textklassifikation

Die Klassifikation ist eine der gängigsten Anwendungen im Bereich des Natural Language Processing. Dabei wird ein Modell so trainiert, dass es einen vorgegebenen Text einer Kategorie bzw. Klasse zuordnen kann (siehe Abbildung 4.1). Die Textklassifikation kommt in einer Vielzahl von Anwendungsbereichen zum Einsatz, von der Sentimentanalyse und der Intentionserkennung (engl. *Intent Detection*) bis hin zur Eigennamenerkennung (engl. *Named-Entity Recognition*, NER) und Spracherkennung. Die Art und Weise, wie Texte heutzutage klassifiziert werden, wurde maßgeblich sowohl von Representation-Modellen als auch von generativen Sprachmodellen beeinflusst.

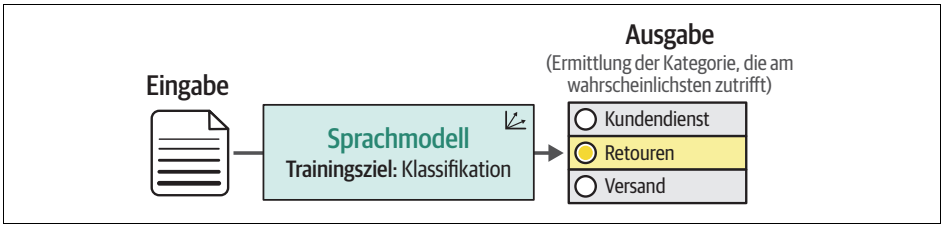

Abbildung 4.1: Sprachmodelle können auch zur Klassifizierung von Texten verwendet werden.

In diesem Kapitel werden Sie mit den verschiedenen Möglichkeiten vertraut gemacht, wie Sprachmodelle zur Klassifizierung von Texten eingesetzt werden können. Dabei sammeln Sie erste Erfahrungen im Umgang mit bereits vortrainierten Sprachmodellen. Da es auf dem Gebiet der Textklassifikation ein breites Spektrum an Möglichkeiten gibt, werden Ihnen zunächst die verschiedenen Methoden vorgestellt. Anschließend können Sie sich anhand dieser Methoden mit den Möglichkeiten vertraut machen, die sich im Zusammenhang mit Sprachmodellen bieten.

- In Abschnitt »Texte mit Representation-Modellen klassifizieren« auf Seite 133 erfahren Sie zunächst, wie flexibel nicht generative Modelle für die Klassifikation genutzt werden können. Dabei werden sowohl aufgabenspezifische Modelle als auch Embedding-Modelle behandelt.
- In Abschnitt »Texte mit generativen Modellen klassifizieren« auf Seite 147 erhalten Sie eine Einführung in generative Sprachmodelle, von denen die meisten zur Klassifikation herangezogen werden können. In diesem Zusammenhang werden sowohl Open-Source- als auch Closed-Source-Sprachmodelle beleuchtet.

In diesem Kapitel geht es in erster Linie darum, wie vortrainierte Sprachmodelle genutzt werden können – also Modelle, die bereits auf der Grundlage großer Datenmengen trainiert wurden und direkt zur Klassifizierung von Texten eingesetzt werden können. Dabei werden Sie sowohl Representation-Modelle als auch generative Sprachmodelle verwenden und deren Unterschiede kennenlernen (siehe Abbildung 4.2).

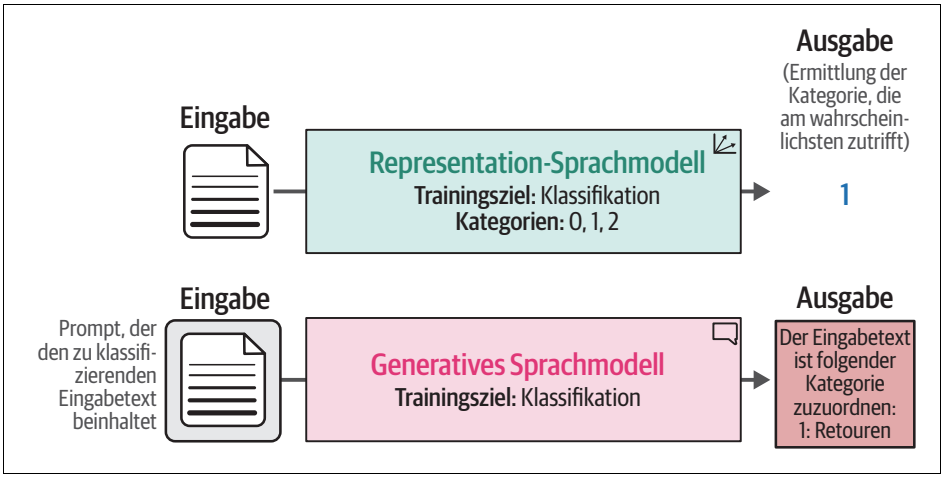

Abbildung 4.2: Sowohl Representation- als auch generative Modelle eignen sich für die Klassifikation. Allerdings unterscheiden sich beide Ansätze voneinander.

Im Folgenden lernen Sie verschiedene generative und nicht generative Sprachmodelle sowie die gängigsten Pakete kennen, mit denen sich diese Modelle laden und verwenden lassen.

Zwar liegt der thematische Schwerpunkt dieses Buchs auf LLMs, dennoch sollten Sie die vorgestellten Beispiele unbedingt mit traditionellen, gleichwohl effektiven Methoden vergleichen. Beispielsweise könnten Sie eine Vektordarstellung für den entsprechenden Text mittels TF-IDF erstellen und darauf aufbauend einen Klassifikator trainieren, der auf der logistischen Regression basiert.

Sentimentanalyse von Spielfilmrezensionen

Alle Daten, die zur Veranschaulichung der verschiedenen Verfahren zur Textklassifikation verwendet werden, finden Sie auf dem Hugging Face Hub. Dies ist eine Plattform, auf der sowohl Modelle als auch Daten (*https://oreil.ly/ndroe*) gehostet werden. Zum Trainieren und Evaluieren der Modelle wird der bekannte `"rotten_tomatoes"`-Datensatz (*https://oreil.ly/44-1y*) verwendet.[1] Dieser enthält 5.331 positive und 5.331 negative Filmrezensionen von der Website *Rotten Tomatoes*.

1 Bo Pang und Lillian Lee. »Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.« *arXiv Preprint cs/0506075* (2005).

Zum Laden der Daten können Sie das `datasets`-Paket verwenden, das im gesamten Buch zum Einsatz kommt:

```
from datasets import load_dataset

# Datensatz laden
data = load_dataset("rotten_tomatoes")
data
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 8530
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 1066
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1066
    })
})
```

Der Datensatz ist in einen Trainings- (*train*), einen Test- (*test*) und einen Validierungsdatensatz (*validation*) unterteilt. In diesem Kapitel verwenden Sie den Trainingsdatensatz, um die Modelle zu trainieren, und den Testdatensatz, um die Ergebnisse zu validieren. Der zusätzliche Validierungsdatensatz dient dazu, im Rahmen der Optimierung der Hyperparameter (Hyperparameter-Tuning) die Generalisierungsfähigkeit eines Modells zu validieren.

Sehen Sie sich nun zunächst ein Beispiel aus dem Trainingsdatensatz an:

```
data["train"][0, -1]
```

```
{'text': ['the rock is destined to be the 21st century\'s new " conan " and that
he\'s going to make a splash even greater than arnold schwarzenegger , jean-claude
van damme or steven segal .',
  'things really get weird , though not particularly scary : the movie is all
portent and no content .'],
 'label': [1, 0]}
```

Diese kurz gefassten Rezensionen sind entweder mit dem Label »positiv« (`1`) oder »negativ« (`0`) gekennzeichnet. Es handelt sich somit um eine binäre Klassifikation des Sentiments bzw. des zum Ausdruck gebrachten Stimmungsbilds.

Texte mit Representation-Modellen klassifizieren

Die Klassifikation mit vortrainierten Representation-Modellen erfolgt im Allgemeinen auf zwei Arten: entweder mit einem aufgabenspezifischen (engl. *Task-specific*) Modell oder einem Embedding-Modell. Bei diesen Modellen handelt es sich, wie bereits im vorherigen Kapitel erläutert, um Modelle, die durch das Feintuning eines Basismodells (bzw. Foundation-Modells) wie BERT für eine bestimmte nachgelagerte Aufgabe konzipiert sind (siehe Abbildung 4.3).

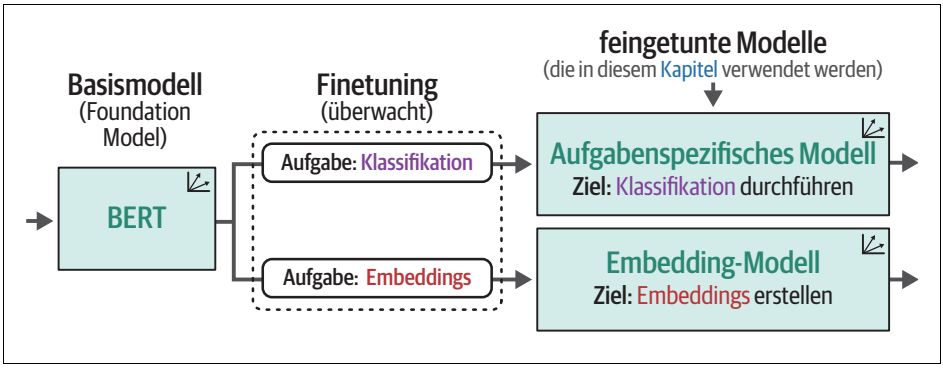

Abbildung 4.3: Ein Foundation-Modell wird für bestimmte Aufgaben feingetunt, beispielsweise für die Klassifikation oder die Generierung von Embeddings, die für verschiedenste Zwecke genutzt werden können.

Ein aufgabenspezifisches Modell ist ein Representation-Modell (wie BERT), das gezielt für eine bestimmte Aufgabe – beispielsweise die Sentimentanalyse – trainiert wurde. Bei einem Embedding-Modell werden, wie Sie in Kapitel 1 erfahren haben, universell einsetzbare Embeddings generiert. Diese können für eine Vielzahl von Aufgaben verwendet werden, also nicht nur für die Klassifikation, sondern auch für andere Aufgaben wie etwa die semantische Suche (siehe Kapitel 8).

In Kapitel 11 erfahren Sie, wie Modelle wie BERT so feingetunt werden können, dass sie sich zur Klassifizierung von Texten eignen. In Kapitel 10 lernen Sie, wie Sie ein eigenes Embedding-Modell erstellen. In diesem Kapitel bleiben beide Modelle *eingefroren* (engl. *frozen*), d. h., sie sind nicht trainierbar, und es wird nur ihre Ausgabe verwendet (siehe Abbildung 4.4).

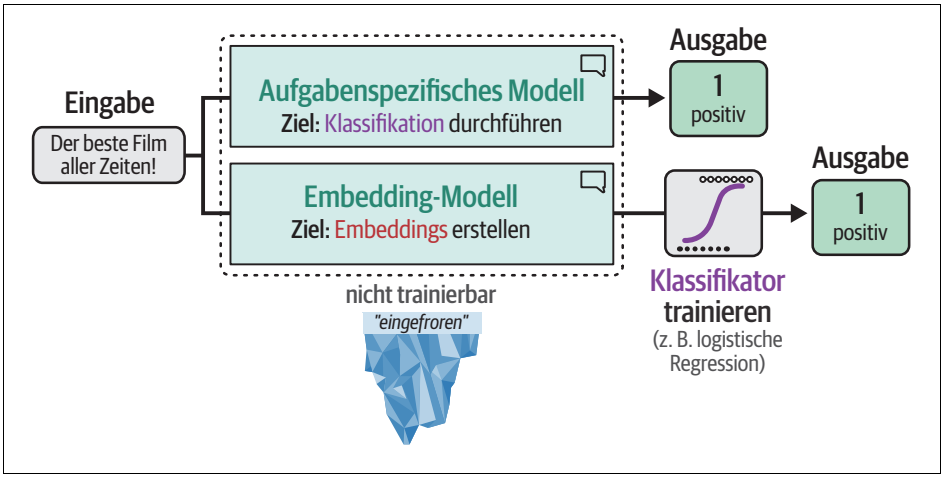

Abbildung 4.4: Eine Klassifikation kann entweder direkt mit einem aufgabenspezifischen Modell oder indirekt mit universell einsetzbaren Embeddings vorgenommen werden.

Zur Klassifizierung der vorliegenden Filmrezensionen greifen Sie nun auf vortrainierte Modelle zurück, die andere bereits für Sie feingetunt haben.

Ein geeignetes Modell auswählen

Angesichts von über 60.000 Modellen auf dem Hugging Face Hub, die sich für die Textklassifikation eignen, (*https://oreil.ly/IPWTY*) und mehr als 8.000 Modellen, die derzeit Embeddings generieren (*https://oreil.ly/yviVH*), ist die Auswahl des richtigen Modells nicht so einfach, wie man vielleicht denken mag. Wichtig ist, dass Sie sich für ein Modell entscheiden, das für Ihren Anwendungsfall geeignet ist. Achten Sie bei der Auswahl darauf, dass es mit der von Ihnen genutzten Sprache kompatibel ist und die entsprechende Architektur, Größe und Leistungsfähigkeit aufweist.

Betrachten wir zunächst die zugrunde liegende Architektur. In Kapitel 1 haben Sie bereits erfahren, dass BERT – eine bekannte, rein auf Encodern basierende Architektur – eine beliebte Wahl ist, wenn es um die Erstellung aufgabenspezifischer Modelle oder Embedding-Modelle geht. Generative Modelle wie die GPT-Modellfamilie mögen zwar beeindruckend sein, doch rein auf Encodern basierende Modelle eignen sich ebenfalls hervorragend für aufgabenspezifische Anwendungsfälle und sind in der Regel deutlich kleiner.

Im Laufe der Jahre wurden etliche Varianten des BERT-Modells entwickelt, wie z. B. RoBERTa[2], DistilBERT[3], ALBERT[4] und DeBERTa[5]. Diese wurden jeweils unter unterschiedlichen Gesichtspunkten trainiert. Abbildung 4.5 hilft Ihnen, sich einen chronologischen Überblick über einige beliebte Varianten des BERT-Modells zu verschaffen.

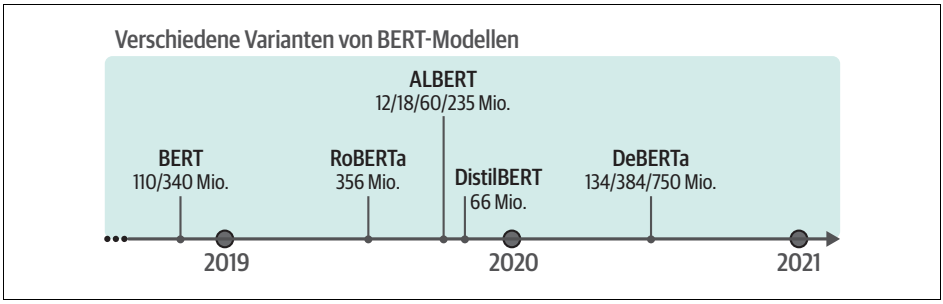

Abbildung 4.5: Eine chronologische Übersicht über bekannte Varianten des BERT-Modells. Hierbei handelt es sich um Basis- bzw. Foundation-Modelle, die hauptsächlich dafür gedacht sind, für eine nachgelagerte Aufgabe feingetunt zu werden.

Das richtige Modell für die jeweilige Aufgabe auszuwählen, kann an sich schon eine Herausforderung sein. Es ist nicht möglich, die vielen Tausend vortrainierten Modelle zu testen, die auf dem Hub von Hugging Face zu finden sind. Daher ist es wich-

2 Yinhan Liuet et al. »RoBERTa: A robustly optimized BERT pretraining approach.« *arXiv Preprint arXiv:1907.11692* (2019).

3 Victor Sanh et al. »DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.« *arXiv Preprint arXiv:1910.01108* (2019).

4 Zhenzhong Lan et al. »ALBERT: A lite BERT for self-supervised learning of language representations.« *arXiv Preprint arXiv:1909.11942* (2019).

5 Pengcheng He et al. »DeBERTa: Decoding-enhanced BERT with disentangled attention.« *arXiv Preprint arXiv:2006.03654* (2020).

tig, mit den Modellen, die man auswählt, möglichst zielführend vorzugehen. Es gibt allerdings mehrere Modelle, die sich hervorragend als Ausgangspunkt eignen und Ihnen einen Eindruck von der grundlegenden Leistungsfähigkeit dieser Art von Modellen vermitteln. Sie sollten sie also als eine solide Ausgangsbasis betrachten:

- BERT-Base-Modell (uncased) (*https://oreil.ly/nq_GM*)
- RoBERTa-Base-Modell (*https://oreil.ly/rz4dQ*)
- DistilBERT-Base-Modell (uncased) (*https://oreil.ly/ieLs3*)
- DeBERTa-Base-Modell (*https://oreil.ly/wN8yl*)
- bert-tiny (*https://oreil.ly/HLRPn*)
- ALBERT-Base-Modell v2 (*https://oreil.ly/Mw93z*)

Im vorliegenden Fall dient das aufgabenspezifische Modell »Twitter-RoBERTa-base for Sentiment Analysis« (*https://oreil.ly/HmvFk*) als Grundlage. Dabei handelt es sich um ein RoBERTa-Modell, das für die Sentimentanalyse von Tweets feingetunt wurde. Zwar wurde das Modell nicht speziell auf Filmrezensionen trainiert, doch es ist durchaus interessant, herauszufinden, wie gut es generalisieren kann.

Wenn Sie auf der Suche nach geeigneten Modellen zur Erzeugung von Embeddings sind, ist das MTEB-Leaderboard (*https://oreil.ly/mUVXD*) ein guter Ausgangspunkt. Dort sind verschiedene Open- und Closed-Source-Modelle aufgeführt, die hinsichtlich mehrerer Aufgaben bewertet wurden. Bei der Auswahl sollten Sie allerdings nicht nur die Leistungsfähigkeit berücksichtigen. Im Hinblick auf die Praxistauglichkeit ist insbesondere die Geschwindigkeit bei der Inferenz ein entscheidender Faktor. Daher wird in diesem Abschnitt »sentence-transformers/all-mpnet-base-v2« (*https://oreil.ly/3pozB*) als Embedding-Modell verwendet. Es handelt sich dabei um ein kleines, aber dennoch recht performantes Modell.

Ein aufgabenspezifisches Modell verwenden

Nachdem Sie sich für ein aufgabenspezifisches Representation-Modell entschieden haben, müssen Sie dieses zunächst vom Hub herunterladen und dann mithilfe einer Pipeline laden:

```
from transformers import pipeline

# Pfad zum Hugging-Face-Modell
model_path = "cardiffnlp/twitter-roberta-base-sentiment-latest"

# Modell in Pipeline laden
pipe = pipeline(
    model=model_path,
    tokenizer=model_path,
    return_all_scores=True,
    device="cuda:0"
)
```

Wenn Sie das Modell laden, wird auch der zugehörige *Tokenizer* geladen, der für die Zerlegung des eingegebenen Texts in einzelne Tokens verantwortlich ist (siehe Abbildung 4.6). Der entsprechende Parameter muss nicht explizit angegeben werden,

da der Tokenizer automatisch geladen wird – er dient hier lediglich der Veranschaulichung dessen, was im Hintergrund geschieht.

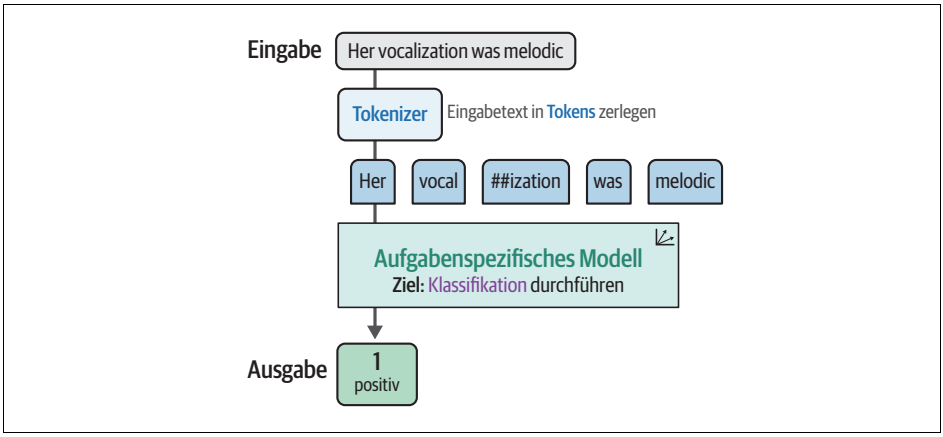

Abbildung 4.6: Bevor ein eingegebener Satz vom aufgabenspezifischen Modell verarbeitet werden kann, wird er zunächst an den Tokenizer geleitet.

Tokens sind – wie bereits in Kapitel 2 ausführlich erläutert – ein integraler Bestandteil der meisten Sprachmodelle. Ein großer Vorteil der Tokenisierung besteht darin, dass Tokens miteinander kombiniert werden können und dass die entsprechenden Vektordarstellungen auch dann erzeugt werden können, wenn die eingegebenen Wörter nicht in den Trainingsdaten enthalten waren (siehe Abbildung 4.7).

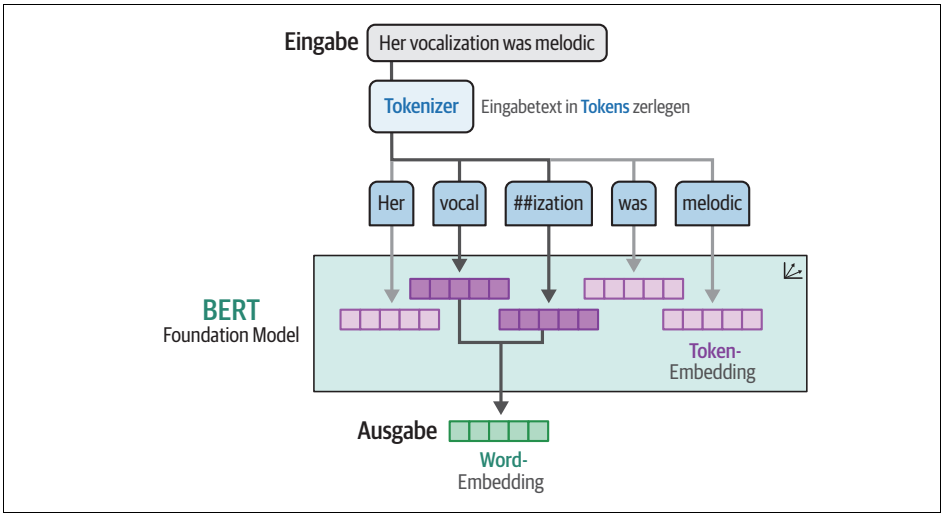

Abbildung 4.7: Selbst wenn ein Wort unbekannt ist bzw. nicht in den Trainingsdaten enthalten war, können in der Regel dennoch entsprechende Word-Embeddings generiert werden, da das Wort zunächst in Tokens zerlegt wird.

Alle erforderlichen Komponenten sind nun eingerichtet. Probieren Sie das Modell doch gleich auf dem Testdatensatz aus:

```
import numpy as np
from tqdm import tqdm
from transformers.pipelines.pt_utils import KeyDataset

# Vorhersagen treffen (Inferenz)
y_pred = []
for output in tqdm(pipe(KeyDataset(data["test"], "text")), total=len(data[
"test"])):
    negative_score = output[0]["score"]
    positive_score = output[2]["score"]
    assignment = np.argmax([negative_score, positive_score])
    y_pred.append(assignment)
```

Liegen Ihnen die Vorhersagen des Modells vor, steht noch deren Evaluierung aus. Hierzu können Sie die folgende Funktion verwenden, die in diesem Kapitel immer wieder zum Einsatz kommen wird:

```
from sklearn.metrics import classification_report

def evaluate_performance(y_true, y_pred):
    """Klassifikationsbericht erstellen und anzeigen lassen"""
    performance = classification_report(
        y_true, y_pred,
        target_names=["Negative Review", "Positive Review"]
    )
    print(performance)
```

Mit dieser Funktion können Sie sich einen Bericht erstellen lassen, anhand dessen Sie beurteilen können, wie gut die Klassifikation des Modells gelungen ist:

```
evaluate_performance(data["test"]["label"], y_pred)
```

```
                precision    recall  f1-score   support

Negative Review       0.76      0.88      0.81       533
Positive Review       0.86      0.72      0.78       533

      accuracy                           0.80      1066
     macro avg       0.81      0.80      0.80      1066
  weighted avg       0.81      0.80      0.80      1066
```

Um die im Bericht aufgeführten Ergebnisse der binären Klassifikation richtig interpretieren und beurteilen zu können, muss zunächst geklärt werden, wie Vorhersagen eingeteilt werden können. Je nachdem, welche Kategorie vorhergesagt wurde (positiv oder negativ) und welche Kategorie tatsächlich zutrifft, gibt es vier mögliche Kombinationen. Wenn es sich um ein Beispiel handelt, das als positiv klassifiziert wurde, kann die Vorhersage entweder korrekt (richtig positiv, engl. *True Positive* – TP) oder falsch (falsch positiv, engl. *False Positive* – FP) sein. Entsprechend kann die Vorhersage im Fall eines Beispiels, das als negativ klassifiziert wurde, ebenfalls entweder korrekt (richtig negativ, engl. *True Negative* – TN) oder falsch (falsch negativ, engl. *False Negative* – FN) sein. Diese Kombinationen können als Matrix dargestellt werden, die im Allgemeinen als Wahrheitsmatrix, teils auch als Konfusionsmatrix (engl. *Confusion Matrix*) bezeichnet wird (siehe Abbildung 4.8).

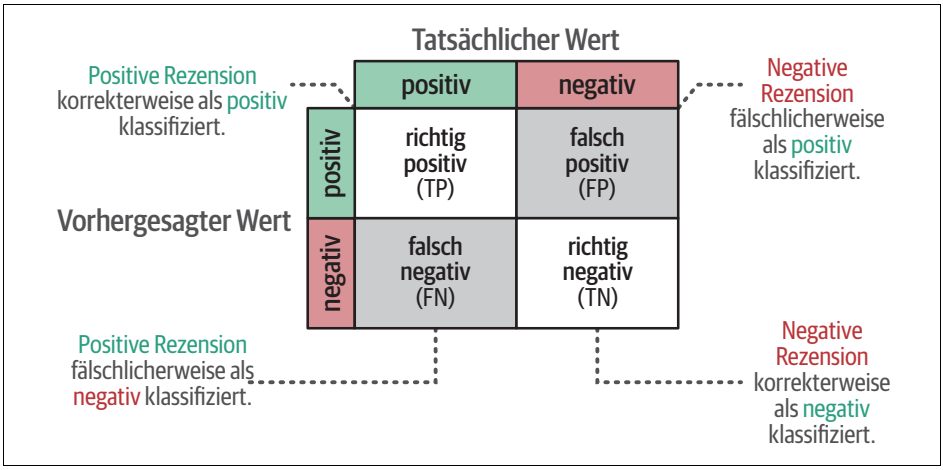

Abbildung 4.8: Die vier verschiedenen Arten von Vorhersagen lassen sich aus der Wahrheitsmatrix ablesen.

Auf der Grundlage der Wahrheitsmatrix können nun mehrere Maße zur Beurteilung der Qualität bzw. Güte des Modells abgeleitet werden. Im zuvor erstellten Klassifikationsbericht werden vier dieser Maße aufgeführt: die *Relevanz*, auch Genauigkeit genannt (engl. *Precision*), die *Trefferquote* bzw. Sensitivität (engl. *Recall*), die *Treffergenauigkeit*, die auch als Korrektklassifikationsrate (engl. *Accuracy*) bezeichnet wird, und das *F1-Maß* (engl. *F1-Score*). (Da in Python die Ergebnisse auf Englisch ausgegeben werden, werden in der folgenden Liste auch die englischen Begriffe für die Maße verwendet, um eine bessere Nachvollziehbarkeit zu gewährleisten.)

- Die *Precision* (bzw. Relevanz) gibt an, wie viele der als relevant (d.h. positiv) klassifizierten Beispiele korrekt klassifiziert wurden, und spiegelt somit die Treffergenauigkeit hinsichtlich der relevanten bzw. positiven Beispiele wider.
- Der *Recall* (bzw. die Trefferquote) gibt an, wie viele der tatsächlich relevanten (d.h. positiven) Beispiele korrekt klassifiziert wurden, und dient somit als Indikator für die Fähigkeit, alle relevanten bzw. positiven Beispiele zu identifizieren.
- *Accuracy* (bzw. Treffergenauigkeit) bezieht sich darauf, wie viele Vorhersagen das Modell in Bezug auf alle Vorhersagen korrekt getroffen hat, und dient dementsprechend als Indikator davon, wie viele Vorhersagen des Modells insgesamt korrekt getroffen wurden.
- Beim *F1-Score* (bzw. F1-Maß) werden sowohl die Precision als auch der Recall (in Form eines gewichteten harmonischen Mittels) berücksichtigt, womit sich beurteilen lässt, wie gut ein Modell insgesamt abschneidet.

In Abbildung 4.9 werden die vier Maße anhand des von Ihnen zuvor erstellten Klassifikationsberichts veranschaulicht.

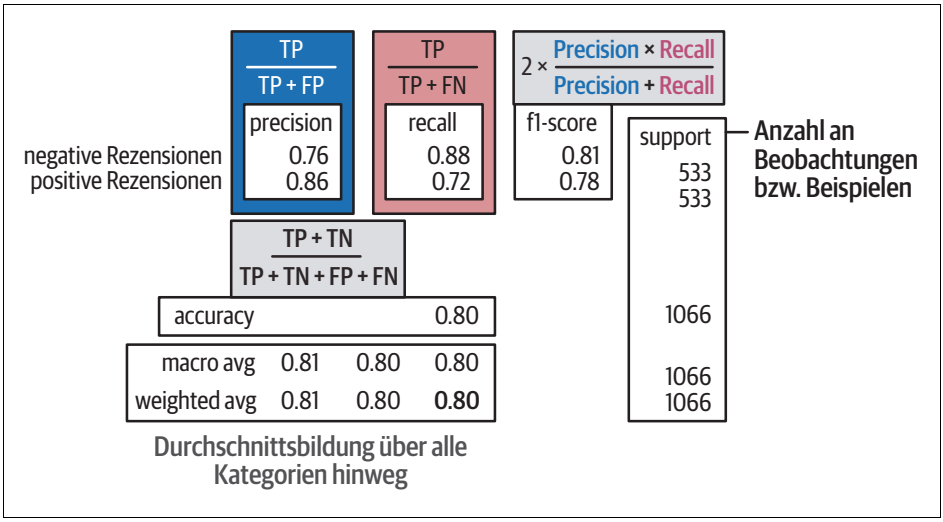

Abbildung 4.9: Der Klassifikationsbericht umfasst mehrere Maße, anhand deren sich die Performance eines Modells bewerten lässt.

In diesem Buch wird bei allen Beispielen eine Form der gewichteten Mittelwertbildung (engl. *Weighted Average*) für das F1-Maß zugrunde gelegt. Dadurch wird sichergestellt, dass alle Kategorien gleichermaßen ins Gewicht fallen (was vor allem bei Datensätzen von Bedeutung ist, bei denen die Kategorien nicht gleichmäßig vertreten sind). Das vortrainierte BERT-Modell liefert einen Wert von 0,80 für das F1-Maß (dieser Wert wird aufgeführt in der Zeile *weighted avg* der Spalte *f1-score*). Angesichts der Tatsache, dass das Modell nicht gezielt im Hinblick auf Daten dieser Domäne trainiert wurde, ist dies ein hervorragendes Ergebnis!

Es gibt eine Reihe von Möglichkeiten, das Modell noch weiter zu verbessern. Sie könnten beispielsweise ein Modell auswählen, das auf Basis von Daten aus der entsprechenden Domäne trainiert wurde – im vorliegenden Fall also auf Basis von Filmrezensionen. Hierfür könnten Sie etwa das Modell DistilBERT base uncased fine-tuned SST-2 (*https://oreil.ly/7-zVj*) verwenden. Sie könnten sich aber auch auf eine andere Art von Representation-Modell stützen, nämlich auf Embedding-Modelle.

Texte mit Embedding-Modellen klassifizieren

Im vorangegangenen Beispiel haben Sie ein vortrainiertes, aufgabenspezifisches Modell eingesetzt, das speziell für die Sentimentanalyse ausgelegt ist. Was aber, wenn Sie kein solches Modell finden? Müssen Sie dann selbst ein Feintuning eines Representation-Modells vornehmen? Die Antwort lautet: Nein!

Wenn Sie über die entsprechenden Rechenkapazitäten verfügen, haben Sie auch die Möglichkeit, das Modell selbst feinzutunen (siehe Kapitel 11). Allerdings hat nicht jeder Zugang zu derart umfangreichen Rechenkapazitäten. Glücklicherweise gibt es in solchen Situationen einen Ansatz, der Abhilfe schafft: universell einsetzbare (*general-purpose*) Embedding-Modelle.

Überwachte Klassifikation

Anstatt direkt ein Representation-Modell für die Klassifikation zu verwenden (wie im vorangegangenen Beispiel), kann auch ein traditionellerer Ansatz gewählt werden, bei dem ein Teil des Trainings selbst durchgeführt wird. Dazu verwenden Sie nun anstelle des Representation-Modells ein Embedding-Modell, mit dessen Hilfe Sie Merkmale (*Features*) extrahieren bzw. generieren. Diese können Sie dann in einen Klassifikator einspeisen und so einen zweistufigen Ansatz verfolgen (siehe Abbildung 4.10).

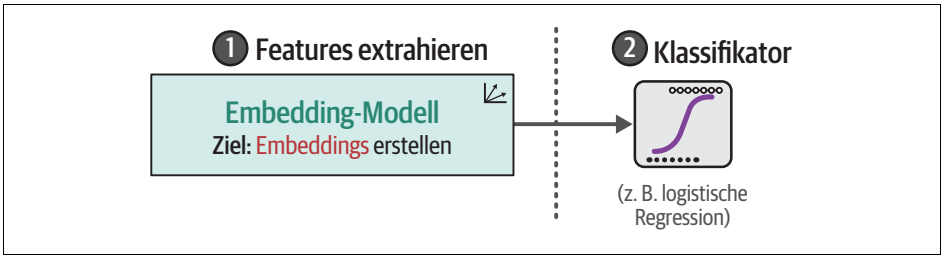

Abbildung 4.10: Die Feature Extraction (erster Schritt) und die Klassifikation (zweiter Schritt) erfolgen separat.

Einer der Hauptvorteile dieses zweistufigen Ansatzes besteht darin, dass kein aufwendiges Feintuning des Embedding-Modells notwendig ist. Stattdessen kann ein Klassifikator, beispielsweise auf Basis einer logistischen Regression, problemlos auf dem Hauptprozessor (CPU) trainiert werden.

Im ersten Schritt wandeln Sie die Texteingabe mithilfe des Embedding-Modells in Embeddings um (siehe Abbildung 4.11). Auch in diesem Beispiel wird das Modell *eingefroren* und während des Trainings nicht aktualisiert.

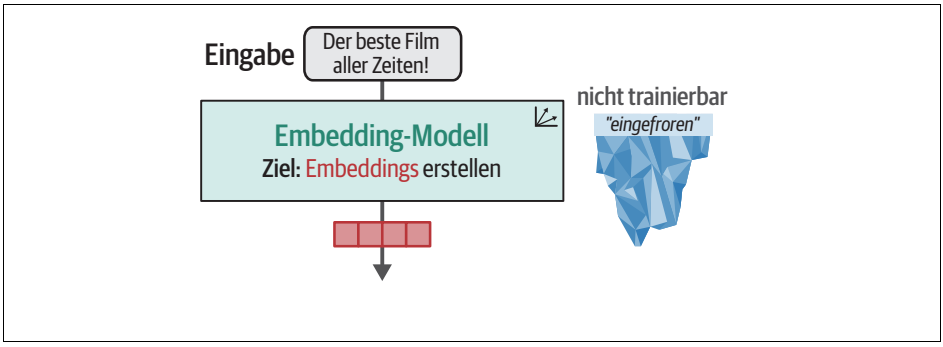

Abbildung 4.11: Im ersten Schritt wird das Embedding-Modell verwendet, um Features zu extrahieren und den Eingabetext in Embeddings umzuwandeln.

Dieser Schritt kann mit dem `sentence-transformers`-Paket durchgeführt werden, das häufig zur Nutzung vortrainierter Embedding-Modelle herangezogen wird.[6] Die Erstellung der Embeddings ist relativ simpel:

6 Nils Reimers und Iryna Gurevych. »Sentence-BERT: Sentence embeddings using Siamese BERT-networks.« *arXiv Preprint arXiv:1908.10084* (2019).

```
from sentence_transformers import SentenceTransformer

# Modell laden
model = SentenceTransformer("sentence-transformers/all-mpnet-base-v2")

# Text in Embeddings umwandeln
train_embeddings = model.encode(data["train"]["text"], show_progress_bar=True)
test_embeddings = model.encode(data["test"]["text"], show_progress_bar=True)
```

Diese Embeddings sind, wie bereits in Kapitel 1 erwähnt, Vektordarstellungen des Eingabetexts. Die Anzahl der Werte bzw. die Anzahl der Dimensionen der Embeddings hängt vom zugrunde liegenden Embedding-Modell ab. Beim vorliegenden Modell sehen diese wie folgt aus:

```
train_embeddings.shape
```

```
(8530, 768)
```

Jedes der 8.530 Eingabedokumente entspricht einem 768-dimensionalen Embedding-Vektor, d. h., ein Embedding umfasst jeweils 768 numerische Werte.

Im zweiten Schritt dienen diese Embeddings als Input-Features für den Klassifikator (siehe Abbildung 4.12). Dieser ist trainierbar und muss nicht zwangsläufig auf Basis einer logistischen Regression trainiert werden. Grundsätzlich ist jede Form eines Klassifikators denkbar, er muss lediglich für eine Klassifikation geeignet sein.

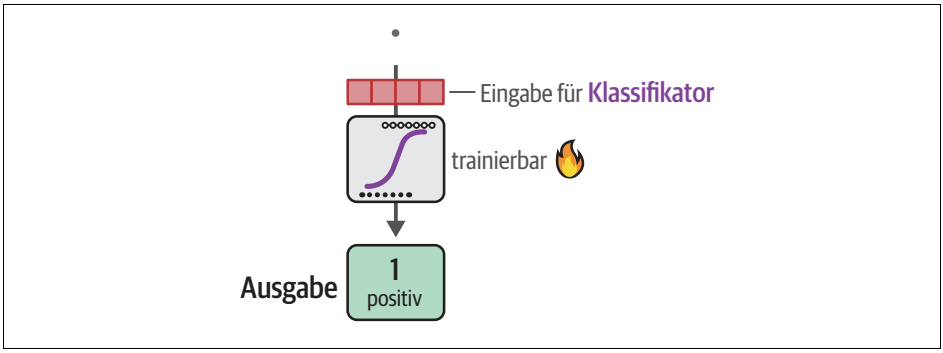

Abbildung 4.12: Auf Basis der als Input-Features verwendeten Embeddings wird ein logistisches Regressionsmodell auf den Trainingsdaten trainiert.

Um diesen Schritt jedoch möglichst einfach zu halten, wird hier als Klassifikator die logistische Regression verwendet. Zum Trainieren des Klassifikators benötigen Sie jetzt lediglich die generierten Embeddings und die entsprechenden Labels:

```
from sklearn.linear_model import LogisticRegression

# Logistisches Regressionsmodell auf Basis der Embeddings,
# die mit den Trainingsdaten erstellt wurden
clf = LogisticRegression(random_state=42)
clf.fit(train_embeddings, data["train"]["label"])
```

Im Anschluss müssen Sie das Modell nur noch evaluieren:

```
# Vorhersagen für den (dem Modell unbekannten) Testdatensatz treffen
y_pred = clf.predict(test_embeddings)
evaluate_performance(data["test"]["label"], y_pred)
```

```
              precision    recall  f1-score   support

Negative Review       0.85      0.86      0.85       533
Positive Review       0.86      0.85      0.85       533

       accuracy                           0.85      1066
      macro avg       0.85      0.85      0.85      1066
   weighted avg       0.85      0.85      0.85      1066
```

Indem Sie einen Klassifikator auf der Grundlage der zuvor erstellten Embeddings trainiert haben, konnten Sie einen Wert für das F1-Maß von 0,85 erzielen. Das zeigt, welche guten Ergebnisse man mit einem relativ simplen Klassifikationsmodell erzielen kann, ohne das zugrunde liegende Embedding-Modell verändern zu müssen bzw. ein weiteres Training vorzunehmen.

Im vorliegenden Beispiel haben Sie zur Extraktion der Embeddings das `sentence-transformers`-Paket verwendet, bei dem die Inferenz auf einer GPU erfolgt und somit wesentlich beschleunigt wird. Sollten Sie keine GPU nutzen können, haben Sie auch die Möglichkeit, eine externe API zur Erstellung der Embeddings zu verwenden. Beispielsweise bieten sowohl Cohere als auch OpenAI Lösungen an, mit denen sich die Pipeline zur Erstellung der Embeddings vollständig auf der CPU ausführen lässt.

Was aber, wenn Ihnen keine gelabelten Daten zur Verfügung stehen?

Im vorangegangenen Beispiel lagen Ihnen bereits gelabelte Daten vor, die Sie für die Klassifikation nutzen konnten. In der Praxis ist dies jedoch nicht immer der Fall. Die Erstellung von gelabelten Daten ist mit einem hohen Aufwand verbunden und erfordert in der Regel viele personelle Ressourcen. Darüber hinaus sollte stets hinterfragt werden, ob die Erstellung dieser Labels auch wirklich erforderlich ist.

Um herauszufinden, ob es sich überhaupt lohnt, Labels zu erstellen, können Sie zunächst eine Zero-Shot-Klassifikation durchführen, also eine Klassifikation ohne gelabelte Daten. Dabei wissen Sie zwar, wie die Labels lauten sollten (ihre Namen), verfügen jedoch nicht über entsprechend gelabelte Daten. Bei der Zero-Shot-Klassifikation wird das Ziel verfolgt, die Labels von Eingabetexten vorherzusagen, ohne dass das System zuvor darauf trainiert wurde (siehe Abbildung 4.13).

Um eine Zero-Shot-Klassifikation auf der Grundlage von Embeddings durchzuführen, bietet sich ein einfacher Trick an: Beschreiben Sie Ihre Labels einfach so, dass bestmöglich wiedergegeben wird, was sie zum Ausdruck bringen sollen. Ein negatives Label für die Rezension eines Spielfilms kann beispielsweise als »Das ist eine Filmrezension, die eine negative Bewertung zum Ausdruck bringt« beschrieben werden. Diese Beschreibungen müssen anschließend in Embeddings umgewandelt werden.

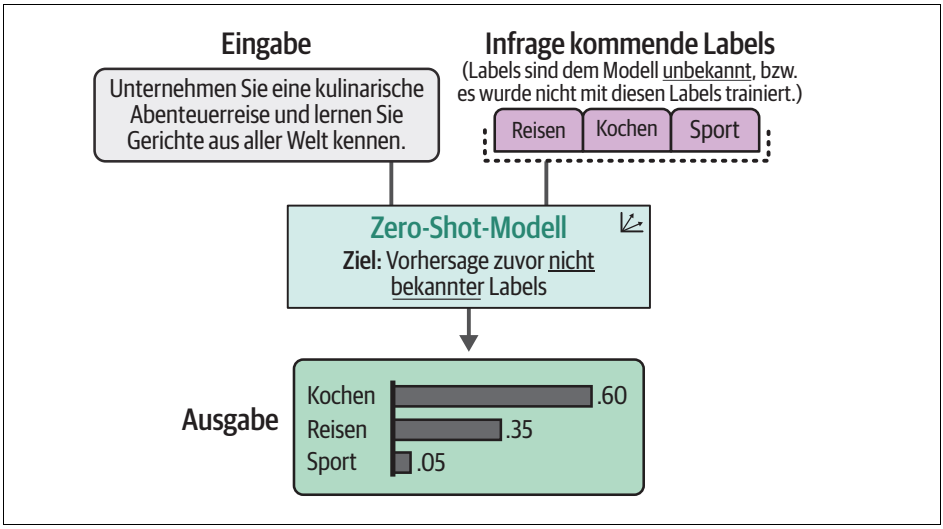

Abbildung 4.13: Bei der Zero-Shot-Klassifikation liegen keine gelabelten Daten vor. Es ist lediglich bekannt, welche Labels bzw. Kategorien in die Klassifikation einbezogen werden sollen. Das Zero-Shot-Modell beurteilt, welchem der infrage kommenden Labels die eingegebenen Daten jeweils zugeordnet werden können.

Zusammen mit den Embeddings der Eingabedokumente können sie dann als Datensatz für die Zero-Shot-Klassifikation genutzt werden. So können Sie Ihre eigenen Target-Labels – also die Labels, die dann zur Klassifikation genutzt werden können – generieren, ohne selbst über gelabelte Daten zu verfügen (siehe Abbildung 4.14).

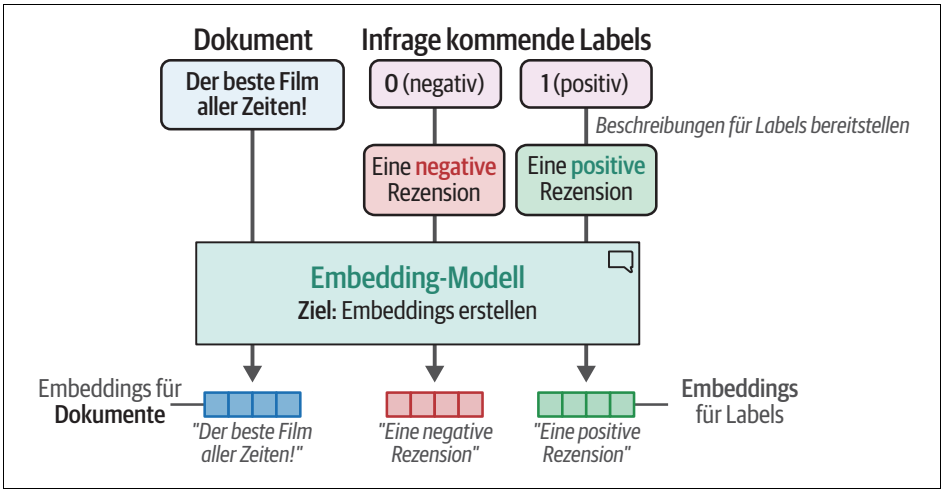

Abbildung 4.14: Um Embeddings für die Labels zu erstellen, ist es zunächst notwendig, deren Bedeutung zu beschreiben, beispielsweise durch Formulierungen wie »Eine negative Rezension« oder »Das ist eine Filmrezension, die eine negative Bewertung zum Ausdruck bringt« (bzw. auf Englisch, wenn die Beispiele des Datensatzes auf Englisch sind). Mithilfe des sentence-transformers-Pakets kann diese Beschreibung dann in ein Embedding umgewandelt werden.

Die Embeddings der Labels erstellen Sie – wie zuvor – mithilfe der Funktion `.encode`:

```
# Embeddings für die Labels erstellen
label_embeddings = model.encode(["A negative review",  "A positive review"])
```

Die Labels können nun mithilfe der Kosinus-Ähnlichkeit (engl. *Cosine Similarity*) den Dokumenten zugeordnet werden. Die Kosinus-Ähnlichkeit zweier Vektoren entspricht dem Kosinus des zwischen ihnen eingeschlossenen Winkels. Zur Berechnung wird zunächst das Skalarprodukt beider Embeddings ermittelt, und dieses wird durch das Produkt ihrer euklidischen Normen (d.h. Längen) dividiert, wie Abbildung 4.15 veranschaulicht.

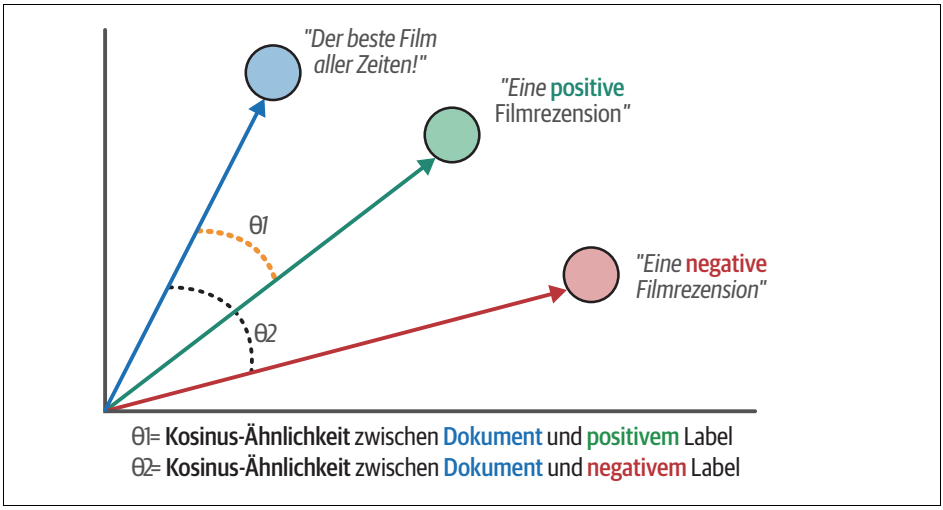

Abbildung 4.15: Die Kosinus-Ähnlichkeit entspricht dem Winkel zwischen zwei Vektoren bzw. Embeddings. Im vorliegenden Beispiel wird jeweils die Ähnlichkeit zwischen einem Dokument und den beiden infrage kommenden Labels (positiv und negativ) ermittelt.

Die Kosinus-Ähnlichkeit gibt Aufschluss darüber, wie ähnlich ein bestimmtes Dokument den Beschreibungen der infrage kommenden Labels ist. Das Label, dessen Beschreibung die größte Ähnlichkeit mit dem Dokument aufweist, wird als Label ausgewählt (siehe Abbildung 4.16).

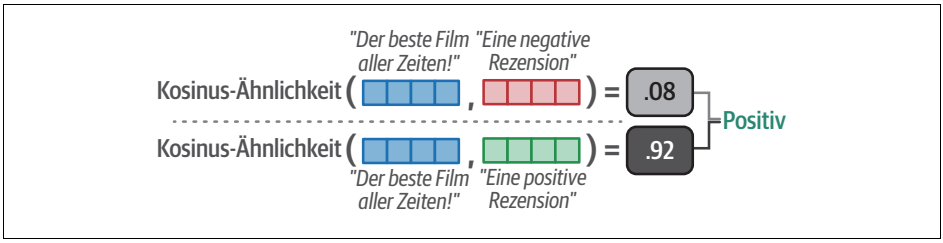

Abbildung 4.16: Nachdem Sie die Embeddings für die Beschreibungen der Labels und die Dokumente erstellt haben, können Sie die Kosinus-Ähnlichkeit zwischen den Labels und den Dokumenten berechnen.

Um nun die Labels zuzuordnen, müssen Sie lediglich die Embeddings der Dokumente mit den Embeddings der Labels auf Basis der Kosinus-Ähnlichkeit vergleichen und ermitteln, welches der Labels die größte Ähnlichkeit mit dem jeweiligen Dokument aufweist:

```
from sklearn.metrics.pairwise import cosine_similarity

# Das Label mit der größten Ähnlichkeit mit dem jeweiligen Dokument ermitteln
sim_matrix = cosine_similarity(test_embeddings, label_embeddings)
y_pred = np.argmax(sim_matrix, axis=1)
```

Das war es bereits! Um die Klassifikation vornehmen zu können, war es im Grunde nur nötig, den Labels eine Bezeichnung zu geben. Mal sehen, wie gut der Ansatz funktioniert:

```
evaluate_performance(data["test"]["label"], y_pred)
```

```
                precision    recall  f1-score   support

Negative Review       0.78      0.77      0.78       533
Positive Review       0.77      0.79      0.78       533

      accuracy                           0.78      1066
     macro avg       0.78      0.78      0.78      1066
  weighted avg       0.78      0.78      0.78      1066
```

Wenn Sie sich bereits mit der Zero-Shot-Klassifikation (*https://oreil.ly/jpayB*) mit Transformer-basierten Modellen auskennen, fragen Sie sich möglicherweise, warum in diesem Buch ein anderer Ansatz, nämlich der mit Embeddings, vorgestellt wird. Zwar eignen sich Natural-Language-Inference-Modelle hervorragend für die Zero-Shot-Klassifikation, doch dieses Beispiel verdeutlicht, wie flexibel einsetzbar Embeddings sind und warum sie für eine Vielzahl unterschiedlichster Aufgaben infrage kommen können. Im weiteren Verlauf des Buchs werden Sie feststellen, dass Embeddings in den meisten Anwendungsfällen im Bereich Language AI zum Einsatz kommen können und eine oftmals unterschätzte, aber ungemein wichtige Rolle spielen.

Wenn man bedenkt, dass überhaupt keine gelabelten Daten vorlagen, ist ein Wert für das F1-Maß von 0,78 ziemlich beeindruckend! Hier zeigt sich, wie vielseitig und nützlich Embeddings sind – vor allem, wenn man bei ihrer Verwendung ein wenig kreativ ist.

Stellen wir Ihre Kreativität mal auf die Probe! Wir haben uns für »A negative/positive review« als Bezeichnung für unsere Labels entschieden, was aber noch Verbesserungspotenzial bietet. Wie wäre es stattdessen mit einer konkreteren und spezifischeren Bezeichnung, die sich stärker auf den vorliegenden Datensatz bezieht, z. B. »A very negative/positive movie review«? So wird im Embedding berücksichtigt, dass es sich um eine Rezension eines Spielfilms handelt. Bei beiden Labels wird der Fokus außerdem etwas stärker auf extreme Ausprägungen gelegt. Probieren Sie diesen Ansatz doch einfach mal aus und finden Sie heraus, wie sich dies auf die Ergebnisse auswirkt!

Texte mit generativen Modellen klassifizieren

Die Klassifikation mit generativen Sprachmodellen – wie den GPT-Modellen von OpenAI – unterscheidet sich von den bisherigen Ansätzen. In diese Modelle wird ein Text (d.h. eine Sequenz) als Eingabe eingespeist, und sie generieren daraufhin ebenfalls einen Text (d.h. eine Sequenz), deshalb werden sie auch als Sequence-to-Sequence-Modelle bezeichnet. Dieser Ansatz unterscheidet sich deutlich von dem des zuvor verwendeten aufgabenspezifischen Modells, bei dem kein Text, sondern eine Kategorie bzw. Klasse in Form eines numerischen Werts ausgegeben wird (siehe Abbildung 4.17).

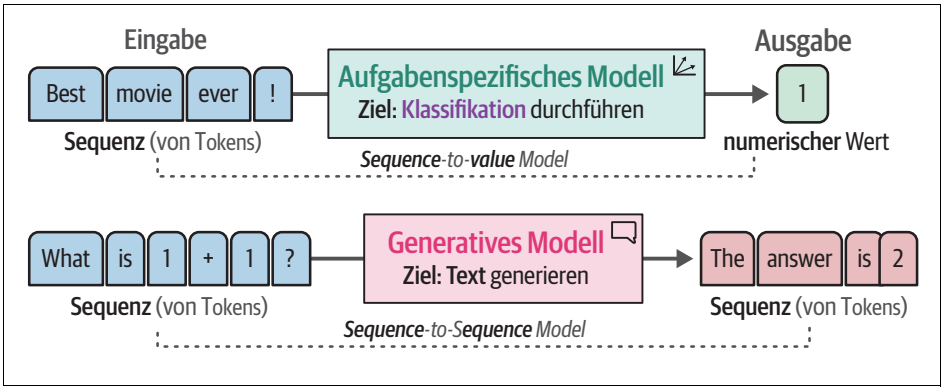

Abbildung 4.17: Ein aufgabenspezifisches Modell gibt infolge einer vorgegebenen Sequenz von Tokens einen numerischen Wert aus (»Sequence-to-Value«-Modell). Ein generatives Modell generiert hingegen infolge einer vorgegebenen Sequenz von Tokens eine weitere Sequenz von Tokens (»Sequence-to-Sequence«-Modell).

Generative Modelle werden in der Regel für ein breites Spektrum an Aufgaben trainiert. In den meisten Fällen deckt dies Ihren Anwendungsfall jedoch nicht von vornherein ab. Wenn man einem solchen Modell beispielsweise eine Rezension zu einem Spielfilm vorgibt, ohne dabei näher auf den Kontext einzugehen, weiß es nicht, was es damit anfangen soll.

Daher ist es notwendig, dem Modell den Kontext zu vermitteln und ihm Hinweise dazu zu geben, welche Art von Antworten erwartet wird. Dies lässt sich am besten in Form von Anweisungen (engl. *Instructions*) bzw. *Prompts* umsetzen, die dem Modell vorgegeben werden (siehe Abbildung 4.18).

Um dem gewünschten Ergebnis nach und nach näherzukommen, müssen die Prompts schrittweise verbessert werden. Das wird als Prompt Engineering bezeichnet.

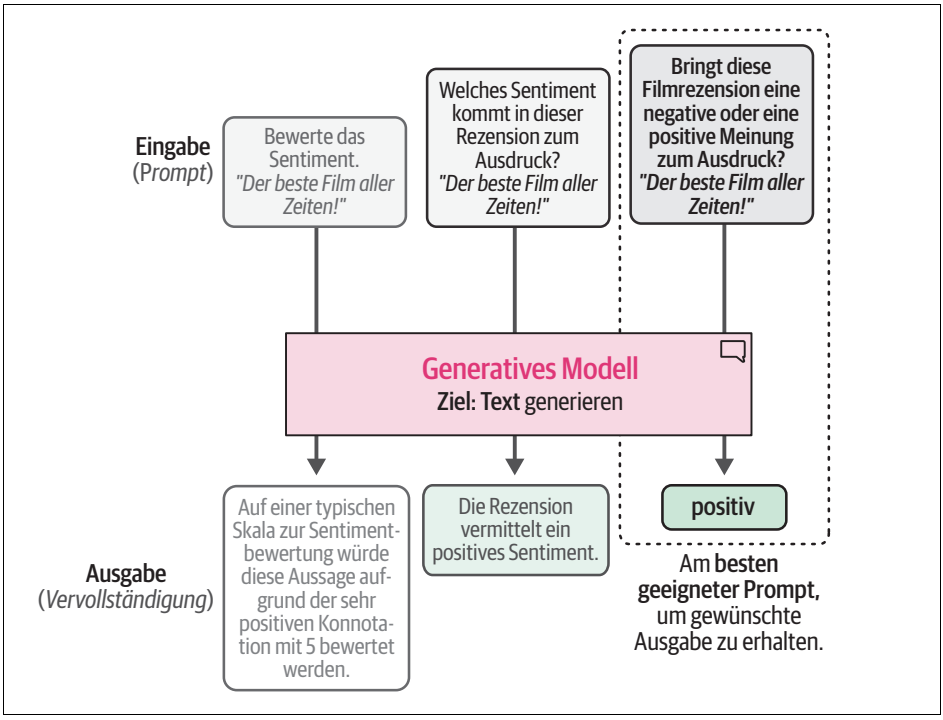

Abbildung 4.18: Im Rahmen des sogenannten Prompt Engineering wird der Prompt nach und nach angepasst, um die vom Modell generierte Ausgabe zu optimieren.

Im folgenden Abschnitt erfahren Sie, wie Sie mithilfe verschiedener Arten generativer Modelle eine Klassifikation vornehmen können – ohne dabei auf den Rotten-Tomatoes-Datensatz als Grundlage zurückzugreifen.

Texte mit T5-(Text-to-Text-Transfer-Transformer-)Modellen klassifizieren

Im gesamten Buch werden hauptsächlich rein auf Encodern basierende Modelle (Representation-Modelle) wie BERT und rein auf Decodern basierende Modelle (generative Modelle) wie ChatGPT verwendet. Bei der ursprünglichen Transformer-Architektur handelt es sich allerdings, wie in Kapitel 1 erläutert, eigentlich um eine Encoder-Decoder-Architektur. Diese Encoder-Decoder-Modelle sind wie die nur aus Decodern bestehenden Modelle ebenfalls Sequence-to-Sequence-Modelle und lassen sich grundsätzlich auch als generative Modelle einordnen.

Eine interessante Modellfamilie, bei der diese Architektur zum Einsatz kommt, basiert auf dem *Text-to-Text Transfer Transformer* – kurz T5. Die Architektur entspricht im Grunde der des ursprünglichen Transformers, bei dem zwölf Decoder und zwölf Encoder hintereinandergeschaltet sind (siehe Abbildung 4.19).[7]

7 Colin Raffel et al. »Exploring the limits of transfer learning with a unified text-to-text transformer.« *The Journal of Machine Learning Research* 21.1 (2020): 5485–5551.

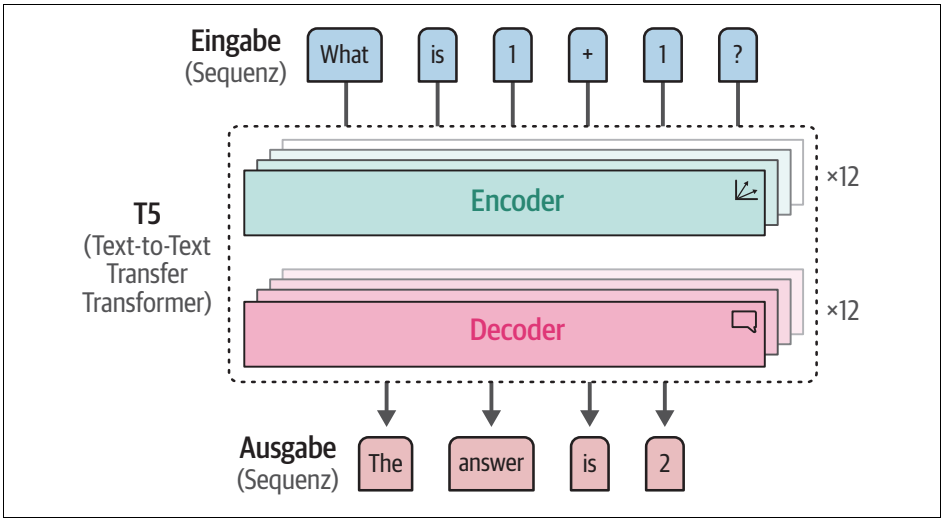

Abbildung 4.19: Die Architektur von T5-Modellen entspricht im Prinzip der des ursprünglichen Transformer-Modells, also einer Decoder-Encoder-Architektur.

Modelle, die auf dieser Architektur basieren, wurden zunächst mithilfe von Masked Language Modeling vortrainiert. Im ersten Schritt des Trainings, der in Abbildung 4.20 illustriert wird, wurden während des Pretrainings nicht einzelne Tokens, sondern mehrere Tokens (engl. *Sets of Tokens* bzw. *Token Spans*) maskiert bzw. verdeckt.

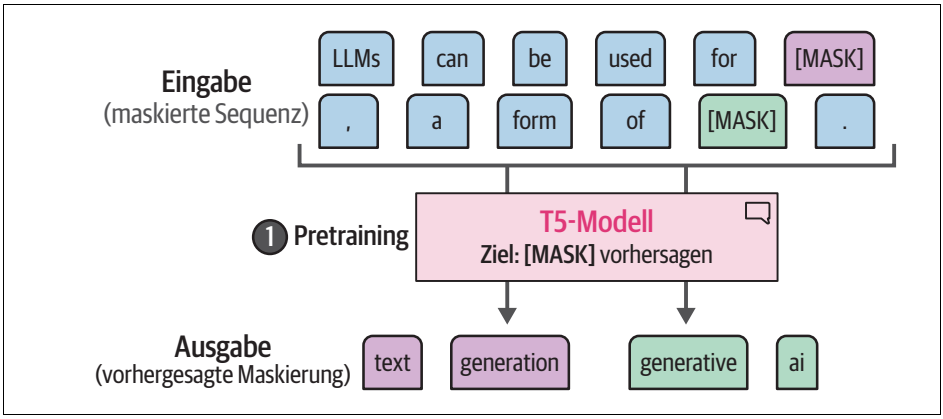

Abbildung 4.20: Im ersten Trainingsschritt, dem Pretraining, muss das T5-Modell maskierte Stellen vorhersagen, an denen sich potenziell mehrere Tokens befinden können.

Im zweiten Trainingsschritt, dem Feintuning des Basismodells, geschieht die eigentliche »Magie«. Anstatt das Modell für eine bestimmte Aufgabe feinzutunen, wird jede Aufgabe in eine Sequence-to-Sequence-Aufgabe umformuliert (d.h. so modifiziert, dass auf eine Eingabe, die eine Sequenz darstellt, eine Ausgabe folgt, die ebenfalls eine Sequenz darstellt). Das Modell wird dann für diese Aufgaben gleichzeitig

trainiert, sodass es für viele Aufgaben eingesetzt werden kann (siehe Abbildung 4.21).

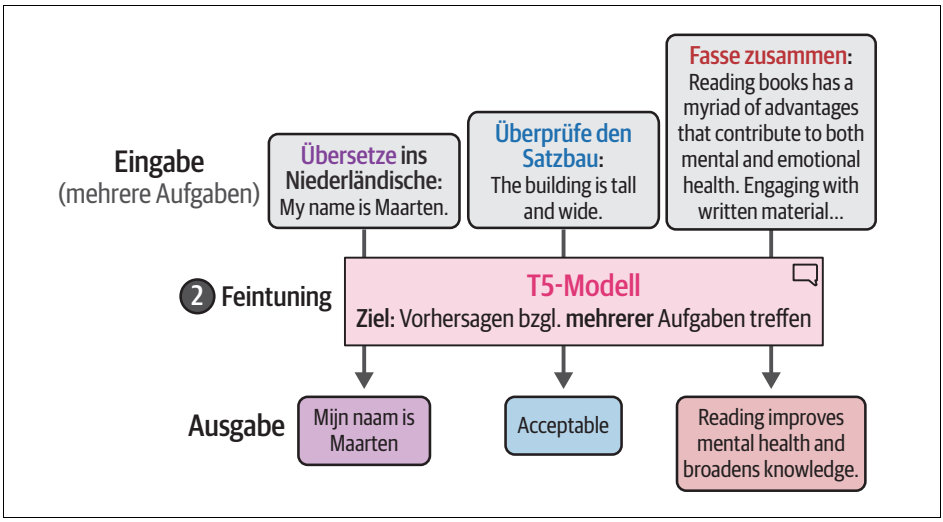

Abbildung 4.21: Da die einzelnen Aufgaben in Anweisungen (in Textform) übertragen werden, kann das T5-Modell während des Feintunings für eine Vielzahl von Aufgaben trainiert werden.

Dieser Ansatz des Feintunings wurde in dem Forschungsartikel »Scaling instruction-finetuned language models« (*https://oreil.ly/yl9Et*) noch weiterentwickelt. Dabei wurden mehr als 1.000 Aufgaben für das Feintuning eingeführt, für die Anweisungen entwickelt wurden, wie man sie von GPT-Modellen kennt.[8] Hieraus ging die Modellreihe Flan-T5 hervor, deren Modelle sich für ein breites Spektrum an Aufgaben eignen.

Wenn Sie dieses vortrainierte Flan-T5-Modell für die Klassifikation verwenden möchten, sollten Sie es zunächst unter Angabe der Aufgabe `"text2text-generation"` laden. Diese ist prinzipiell für derartige Encoder-Decoder-Modelle vorgesehen:

```
# Unser Modell laden
pipe = pipeline(
    "text2text-generation",
    model="google/flan-t5-small",
    device="cuda:0"
)
```

Das Flan-T5-Modell steht in verschiedenen Größen zur Verfügung (`flan-t5-small/base/large/xl/xxl`). Der Einfachheit halber wird im vorliegenden Beispiel das kleinste Modell verwendet. Sie können es aber auch gern mit den größeren Modellen ausprobieren und testen, ob Sie damit noch bessere Ergebnisse erzielen.

8 Hyung Won Chung et al. »Scaling instruction-finetuned language models.« *arXiv Preprint arXiv:2210.11416* (2022).

Anders als beim aufgabenspezifischen Modell können Sie dem Modell nicht einfach einen beliebigen Text vorgeben und hoffen, dass es das zum Ausdruck gebrachte Stimmungsbild (bzw. das Sentiment) erkennt. Stattdessen ist es erforderlich, das Modell entsprechend anzuweisen.

Hierzu bietet es sich an, jedes Dokument mit der Frage »Is the following sentence positive or negative?« zu versehen:

```
# Daten vorbereiten
prompt = "Is the following sentence positive or negative? "
data = data.map(lambda example: {"t5": prompt + example['text']})
data
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 't5'],
        num_rows: 8530
    })
    validation: Dataset({
        features: ['text', 'label', 't5'],
        num_rows: 1066
    })
    test: Dataset({
        features: ['text', 'label', 't5'],
        num_rows: 1066
    })
})
```

Nachdem Sie die Daten entsprechend angepasst haben, können Sie die Pipeline ausführen. Das Vorgehen ist ähnlich wie im Beispiel mit dem aufgabenspezifischen Modell:

```
# Vorhersagen treffen (Inferenz)
y_pred = []
for output in tqdm(pipe(KeyDataset(data["test"], "t5")), total=len(data["test"])):
    text = output[0]["generated_text"]
    y_pred.append(0 if text == "negative" else 1)
```

Da dieses Modell Text generiert, war es zusätzlich notwendig, die Textausgabe in numerische Werte umzuwandeln. Dem Wort »negativ« wurde der Wert 0 und dem Wort »positiv« der Wert 1 zugeordnet.

Da Ihnen nun numerische Werte vorliegen, können Sie die Qualität des Modells auf die gleiche Weise wie zuvor beurteilen:

```
evaluate_performance(data["test"]["label"], y_pred)
```

```
                precision    recall  f1-score   support

Negative Review       0.83      0.85      0.84       533
Positive Review       0.85      0.83      0.84       533

      accuracy                           0.84      1066
     macro avg       0.84      0.84      0.84      1066
  weighted avg       0.84      0.84      0.84      1066
```

Angesichts eines Werts des F1-Maßes von 0,84 haben Sie nun also bereits einen sehr guten Eindruck davon gewinnen können, zu welch erstaunlichen Fähigkeiten generative Modelle wie das Flan-T5-Modell imstande sind.

Texte mit ChatGPT klassifizieren

Obwohl der Schwerpunkt des Buchs auf Open-Source-Modellen liegt, sind Closed-Source-Modelle wie ChatGPT ebenfalls von großer Bedeutung im Bereich der Language AI.

Zwar wurde die zugrunde liegende Architektur des Ursprungsmodells von ChatGPT (GPT-3.5) nicht offengelegt, doch der Name des Modells lässt darauf schließen, dass es sich um eine rein auf Decodern basierende Architektur handelt, wie Sie sie bereits von den zuvor vorgestellten GPT-Modellen kennen.

OpenAI hat jedoch erfreulicherweise einen Überblick über das zugrunde liegende Trainingsverfahren (*https://oreil.ly/-yf84*) veröffentlicht und dabei einen wichtigen Ansatz offengelegt, der als Preference Tuning bezeichnet wird. Bei diesem Verfahren wurden zunächst manuell Daten erstellt, aus denen hervorgeht, welche Ausgaben für bestimmte Input-Prompts erwartet werden (sogenannte Instruktionsdaten). Diese Daten dienten dann als Grundlage für die Erstellung einer ersten Modellvariante (siehe Abbildung 4.22).

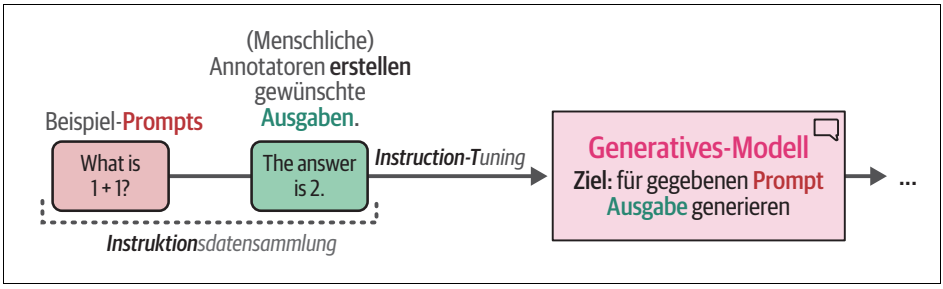

Abbildung 4.22: Für das Feintuning (Instruction Tuning) wurden manuell gelabelte Daten verwendet. Diese bestehen jeweils aus einer Anweisung (Prompt) und der entsprechenden gewünschten Ausgabe.

Mit dem resultierenden Modell wurden daraufhin jeweils mehrere Ausgaben generiert, die anschließend manuell in Form einer Rangfolge bewertet wurden. Aus dieser geht hervor, welche Ausgabe jeweils am besten und welche am schlechtesten angesehen wurde. Die Rangfolge gab somit Aufschluss darüber, welche Ausgaben jeweils präferiert wurden (siehe Abbildung 4.23). Auf Basis dieser Präferenzdaten wurde schließlich das endgültige Modell – ChatGPT – erstellt.

Im Vergleich zu Instruktionsdaten haben Präferenzdaten den großen Vorteil, dass sie dem Modell ein differenzierteres Bild vermitteln. Da das generative Modell lernt, was eine gute und was eine noch bessere Ausgabe ausmacht, ist es in der Lage, Texte zu generieren, die eher den Erwartungen von Menschen entsprechen. In Kapitel 12

erfahren Sie noch genauer, wie solche Feintuning- und Preference-Tuning-Methoden funktionieren und wie Sie diese selbst anwenden können.

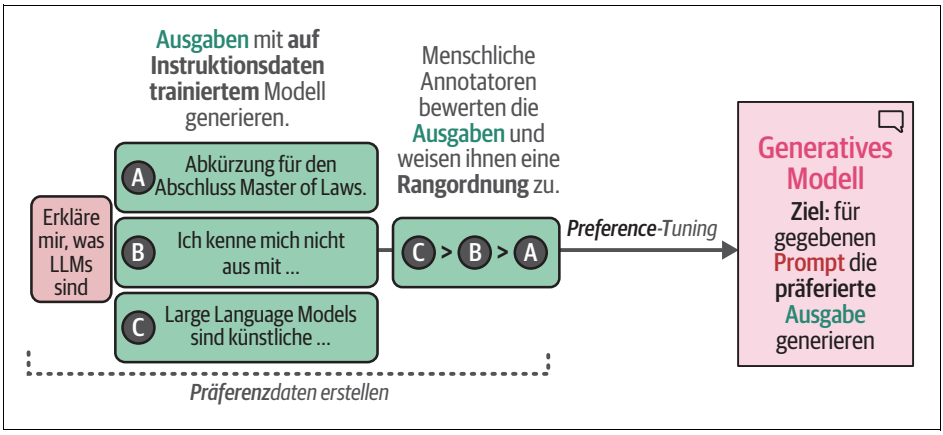

Abbildung 4.23: Zur Erstellung des endgültigen ChatGPT-Modells wurden Präferenzdaten verwendet, also von Menschen bewertete Daten, aus denen hervorging, welche Ausgaben jeweils bevorzugt wurden.

Der Einsatz eines Closed-Source-Modells unterscheidet sich deutlich von den bisher vorgestellten Open-Source-Beispielen. Hier wird das Modell nicht geladen, sondern über die Programmierschnittstelle (API) von OpenAI aufgerufen.

Bevor Sie sich dem Klassifikationsbeispiel zuwenden können, müssen Sie zunächst ein kostenloses Nutzerkonto erstellen (*https://oreil.ly/AEXvA*) und einen API-Schlüssel generieren (*https://oreil.ly/lrTXl*). Anschließend können Sie Ihre API zur Kommunikation mit den Servern von OpenAI nutzen.

Mit dem API-Schlüssel legen Sie anschließend einen Client an:

```
import openai

# Client anlegen
client = openai.OpenAI(api_key="YOUR_KEY_HERE")
```

Sie können nun mithilfe dieses Clients eine Funktion namens `chatgpt_generation` erstellen. Mit dieser Funktion generieren Sie einen Text, indem Sie einen Prompt, ein Eingabedokument und das ausgewählte Modell (wobei `gpt-3.5-turbo-0125` vorausgewählt ist) angeben:

```
def chatgpt_generation(prompt, document, model="gpt-3.5-turbo-0125"):
    """Basierend auf einem Prompt und einem Eingabedokument eine Ausgabe
       generieren."""
    messages=[
        {
            "role": "system",
            "content": "You are a helpful assistant."
            },
        {
            "role": "user",
            "content":   prompt.replace("[DOCUMENT]", document)
```

```
        }
    ]
    chat_completion = client.chat.completions.create(
      messages=messages,
      model=model,
      temperature=0
    )
    return chat_completion.choices[0].message.content
```

Als Nächstes müssen Sie ein sogenanntes Prompt-Template (also eine Vorlage) erstellen, in dem Sie das Modell dazu auffordern, eine Klassifikation vorzunehmen:

```
# Prompt-Template als Ausgangsbasis anlegen
prompt = """Predict whether the following document is a positive or negative movie
review:

[DOCUMENT]

If it is positive return 1 and if it is negative return 0. Do not give any other
answers.
"""

# Vorhersage mit GPT treffen
document = "unpretentious , charming , quirky , original"
chatgpt_generation(prompt, document)
```

Dieses Template dient lediglich als Beispiel und kann von Ihnen nach Belieben geändert werden. Zur Veranschaulichung der Verwendung eines solchen Templates wurde es vorerst so einfach wie möglich gehalten.

Sie sollten sich jedoch zunächst immer ein Bild davon machen, in welchem Umfang die API von Ihnen in Anspruch genommen wird, und sie nicht gleich bedenkenlos für umfangreiche Datensätze nutzen. Der Einsatz externer APIs wie der von OpenAI kann sich nämlich mit zunehmender Anzahl an Anfragen als ziemlich kostspielig erweisen. Derzeit fallen für die Auswertung unseres Testdatensatzes mit dem Modell `gpt-3.5-turbo-0125` Kosten in Höhe von 3 US-Cent an. Diese sind allerdings im Rahmen des kostenfreien Nutzerkontos abgedeckt. Es ist jedoch möglich, dass sich diesbezüglich in Zukunft Änderungen ergeben.

Wenn Sie externe APIs verwenden, können sich Fehler im Zusammenhang mit der Begrenzung der Nutzungsrate (sogenannte *Rate Limit Errors*) ergeben. Diese treten auf, wenn Sie die API innerhalb einer bestimmten Zeit zu oft aufrufen, da bei einigen APIs die Anzahl möglicher Aufrufe, die pro Minute bzw. pro Stunde getätigt werden können, begrenzt ist.

Es gibt mehrere Methoden, um eine solche Fehlermeldung zu verhindern und die Anfrage erneut zu stellen. Eine davon ist das sogenannte *Exponential Backoff*. Dabei wird das Programm jedes Mal, wenn aufgrund einer Ratenbegrenzung ein Fehler auftritt, kurz pausiert, und die fehlgeschlagene Anfrage wird dann erneut gesendet. Schlägt die Anfrage erneut fehl, wird die Pause verlängert, bis die Anfrage erfolgreich ist oder die maximale Anzahl an Wiederholungen erreicht ist.

Es gibt eine sehr gute Anleitung (*https://oreil.ly/ZH4Uo*), die Ihnen den Einstieg in die Nutzung der API von OpenAI erleichtert.

Als Nächstes können Sie die Funktion für alle Spielfilmrezensionen im Testdatensatz ausführen und erhalten so die entsprechenden Vorhersagen. Wenn Sie sich Ihre (kostenfreien) Credits lieber für andere Aufgaben aufsparen möchten, können Sie diesen Schritt auch einfach überspringen.

```
# Sie können diesen Schritt überspringen, wenn Sie sich Ihre (kostenfreien)
# Credits aufsparen möchten.
predictions = [
    chatgpt_generation(prompt, doc) for doc in tqdm(data["test"]["text"])
]
```

Wie im vorherigen Beispiel müssen Sie auch hier die ausgegebenen Zeichenketten in Ganzzahlen konvertieren, bevor Sie die Performance des Modells evaluieren können:

```
# Vorhersagen extrahieren
y_pred = [int(pred) for pred in predictions]

# Performance des Modells evaluieren
evaluate_performance(data["test"]["label"], y_pred)
```

```
                precision    recall  f1-score   support

Negative Review       0.87      0.97      0.92       533
Positive Review       0.96      0.86      0.91       533

      accuracy                           0.91      1066
     macro avg       0.92      0.91      0.91      1066
  weighted avg       0.92      0.91      0.91      1066
```

Ein F1-Maß-Wert von 0,91 für das Modell zeigt, welch leistungsfähige Modelle durch generative KI der breiten Masse zugänglich gemacht wurden. Diese Art von Qualitätsmaßen ist jedoch mit Vorsicht zu genießen, da sich nicht nachvollziehen lässt, mit welchen Daten das Modell trainiert wurde. Es ist durchaus denkbar, dass das Modell mit dem vorliegenden Datensatz trainiert wurde!

In Kapitel 12 erfahren Sie, wie Sie sowohl Open-Source- als auch Closed-Source-Modelle hinsichtlich ihrer Eignung für allgemeinere Aufgaben evaluieren können.

Zusammenfassung

In diesem Kapitel haben Sie verschiedene Ansätze für die Klassifikation kennengelernt – vom Feintuning Ihres gesamten Modells bis hin zum kompletten Verzicht darauf. Die Klassifikation von Textdaten ist nicht so einfach, wie man vielleicht meinen könnte, und erfordert ein gewisses Maß an Kreativität.

In diesem Kapitel wurden sowohl generative als auch Representation-Modelle zur Klassifizierung von Texten herangezogen. Das Ziel bestand darin, dem Eingabetext ein Label bzw. eine Kategorie zuzuweisen und somit das Stimmungsbild bzw. das Sentiment von Rezensionen zu Spielfilmen zu klassifizieren.

Dazu wurden zwei Arten von Representation-Modellen verwendet: ein aufgabenspezifisches Modell und ein Embedding-Modell. Das aufgabenspezifische Modell wurde speziell für die Sentimentanalyse auf einem großen Datensatz vortrainiert. Dies ver-

deutlichte, dass sich vortrainierte Modelle hervorragend zur Klassifizierung von Dokumenten eignen. Das Embedding-Modell wurde zur Generierung universell einsetzbarer Embeddings genutzt, die wiederum als Eingabedaten im Rahmen des Trainings eines Klassifikators dienten.

Zudem haben Sie zwei Arten von generativen Modellen ausprobiert: ein Open-Source-Modell mit einer Encoder-Decoder-Architektur (Flan-T5) und ein Closed-Source-Modell mit einer rein auf Decodern basierenden Architektur (GPT-3.5). Beide generativen Modelle wurden zur Klassifizierung von Texten verwendet, ohne dass ein spezifisches (d. h. zusätzliches) Training anhand von Daten aus der entsprechenden Domäne oder gelabelten Datensätzen erforderlich war.

Im nächsten Kapitel befassen wir uns weiterhin mit der Textklassifikation, widmen uns nun jedoch der unüberwachten Klassifikation. Welche Möglichkeiten gibt es, wenn Textdaten vorliegen, die nicht mit Labels versehen sind? Welche Informationen können daraus gewonnen werden? Dabei stehen vor allem Verfahren zum Clustering von Textdaten und zur Bezeichnung von Clustern mithilfe des Topic Modeling (Themenmodellierung) im Vordergrund.

Inhalt

Vorwort 15

Teil I: Die Funktionsweise von Sprachmodellen verstehen 25

1 Einführung in Large Language Models 27
Was ist Language AI? 28
Die jüngsten Entwicklungen im Bereich der Language AI 29
Darstellung von Sprache als ein Bag-of-Words 30
Verbesserte Vektordarstellungen mit Dense Vector Embeddings . . . 32
Arten von Embeddings 34
Kontext mit Attention codieren und decodieren 35
Attention Is All You Need 38
Representation-Modelle – rein Encoder-basierte Modelle (»Encoder-only«) 41
Generative Modelle – rein Decoder-basierte Modelle (»Decoder-only«) 44
Das Jahr der generativen KI 46
Die sich wandelnde Definition von Large Language Models 48
Wie sich das Training von Large Language Models im Vergleich zu traditionellen Ansätzen unterscheidet 48
Anwendungsmöglichkeiten und Nutzen von Large Language Models . . . 50
LLMs verantwortungsvoll entwickeln und nutzen 51
Limited Resources Are All You Need – LLMs auch ohne große Rechenressourcen trainieren und verwenden 52
Schnittstellen zur Nutzung von LLMs 53
Proprietäre, nicht frei zugängliche Modelle 53
Frei zugängliche Modelle 54
Open-Source-Frameworks 55
Ihren ersten Text mit einem LLM generieren 56
Zusammenfassung 59

2 Tokens und Embeddings 61
Tokenisierung bei LLMs 62
Wie Tokenizer die Eingaben für das Sprachmodell aufbereiten 62
LLMs herunterladen und ausführen 63
Welche Faktoren sind bei der Tokenisierung entscheidend? 67
Tokenisierung auf Wort-, Wortteil-, Zeichen- und Byte-Ebene 68
Ein Vergleich verschiedener trainierter Tokenizer 70
Faktoren, die darüber entscheiden, wie sich ein Tokenizer verhält . . . 78
Token-Embeddings 80
Sprachmodelle halten Embeddings für das Vokabular ihres Tokenizers vor 81
Kontextualisierte Word-Embeddings mit Sprachmodellen erstellen 82
Text-Embeddings (für Sätze oder ganze Dokumente) 85
Wie Word-Embeddings jenseits von LLMs genutzt werden können 86
Vortrainierte Word-Embeddings nutzen 86
Der Word2vec-Algorithmus und Training mittels Contrastive Learning 87
Empfehlungssysteme aufbauen, die auf Embeddings basieren 91
Songs mithilfe von Embeddings empfehlen 91
Embedding-Modelle zur Empfehlung von Songs trainieren 92
Zusammenfassung 94

3 Ein Blick ins Innere von Large Language Models 97
Ein erster Überblick über Transformer-Modelle 98
Die Ein- und Ausgaben eines Transformer-basierten LLM 98
Die einzelnen Komponenten, die beim Forward-Pass durchlaufen werden 100
Auswahl eines einzelnen Tokens anhand einer Wahrscheinlichkeitsverteilung (Sampling/Decodierung) 103
Parallele Verarbeitung von Tokens und Kontextlänge 105
Schnellere Generierung durch Zwischenspeichern von Schlüsseln und Werten 107
Ein Blick ins Innere des Transformer-Blocks 109
Verbesserungen an der Transformer-Architektur 118
Effizienterer Attention-Mechanismus 118
Der Transformer-Block 122
Positional-Embeddings (RoPE) 124
Weitere Vorschläge und Verbesserungen im Hinblick auf die Architektur 127
Zusammenfassung 127

Teil II: Vortrainierte Sprachmodelle verwenden **129**

4 Textklassifikation **131**
Sentimentanalyse von Spielfilmrezensionen 132
Texte mit Representation-Modellen klassifizieren 133
Ein geeignetes Modell auswählen 135
Ein aufgabenspezifisches Modell verwenden 136
Texte mit Embedding-Modellen klassifizieren 140
Überwachte Klassifikation 141
Was aber, wenn Ihnen keine gelabelten Daten zur Verfügung stehen? 143
Texte mit generativen Modellen klassifizieren 147
Texte mit T5-(Text-to-Text-Transfer-Transformer-)Modellen klassifizieren 148
Texte mit ChatGPT klassifizieren 152
Zusammenfassung 155

5 Clustering von Texten und Topic Modeling **157**
ArXiv-Artikel aus dem Forschungsbereich Computation and Language 158
Die bewährte Pipeline beim Text-Clustering 159
Dokumente in Embeddings umwandeln 159
Die Dimensionierung der Embeddings verringern 160
Die dimensionsreduzierten Embeddings zu Clustern zusammenfassen 163
Die gebildeten Cluster inspizieren 164
Vom Clustern von Texten hin zum Topic Modeling 167
BERTopic: ein modulares Topic-Modeling-Framework 168
Der modulare Aufbau des BERTopic-Frameworks 172
Einen besonderen Baustein hinzufügen 176
Ein zusätzlicher Baustein zur Textgenerierung 181
Zusammenfassung 185

6 Prompt Engineering **187**
Textgenerierungsmodelle verwenden 187
Ein geeignetes Textgenerierungsmodell wählen 187
Textgenerierungsmodelle laden 188
Einfluss auf die Ausgabe eines Modells nehmen 190
Einführung in das Prompt Engineering 193
Die grundlegenden Elemente eines Prompts 194
Prompts formulieren, die Anweisungen enthalten (Instruction-based Prompting) 196

Fortgeschrittene Prompt-Engineering-Techniken 198
Prompts komplexer gestalten 198
In-Context Learning – Beispiele bereitstellen 201
Prompt Chaining – Aufgaben in mehrere Teilaspekte aufteilen 203
Logisches Schließen mit generativen Modellen (Reasoning) 205
Chain-of-Thought – erst nachdenken, dann antworten 206
Self-Consistency – Auswahl aus mehreren Antwortmöglichkeiten . . . 209
Tree-of-Thought – über Zwischenschritte zur besten Antwort gelangen 210
Modellausgaben validieren 213
Beispiele bereitstellen 214
Vorgaben machen und Modellausgaben beschränken 216
Zusammenfassung 220

7 Fortgeschrittene Techniken und Tools im Bereich der Textgenerierung 221
Optimierungen hinsichtlich der Verwendung von Modellen – quantisierte Modelle mit dem LangChain-Framework laden 222
Chains – die Anwendungsmöglichkeiten von LLMs noch erweitern 225
Prompt-Templates mit einem LLM verketten 225
Mehrere Prompts miteinander verketten 228
Einen Speicher bereitstellen – LLMs ermöglichen, sich an Gespräche zu erinnern 231
Conversation Buffer – LLMs den gesamten Gesprächsverlauf bereitstellen 233
Windowed Conversation Buffer – LLMs einen Teil des Gesprächsverlaufs bereitstellen 235
Conversation Summary – LLMs eine Zusammenfassung des Gesprächsverlaufs bereitstellen 236
Agenten – ein aus mehreren LLMs bestehendes System entwickeln 240
Die treibende Kraft hinter Agenten – Schritt-für-Schritt-Reasoning 242
ReAct im LangChain-Framework verwenden 244
Zusammenfassung 247

8 Semantische Suche und Retrieval-Augmented Generation 249
Einführung in semantische Such- und RAG-Systeme 250
Semantische Suche auf Basis von LLMs 252
Dense-Retrieval-Systeme 252
Reranking-Systeme 264
Metriken zur Evaluierung von Retrieval-Systemen 268
Retrieval-Augmented Generation (RAG) 273
Suchsysteme zu RAG-Systemen erweitern 274
Ein Beispiel für eine auf Fakten basierende Generierung (Grounded Generation) mit einem gemanagten LLM 276

Ein Beispiel für ein RAG-System, bei dem das Modell lokal betrieben wird . . . 276
Fortgeschrittene Techniken im Bereich der RAG-Systeme . . . 279
RAG-Systeme evaluieren . . . 281
Zusammenfassung . . . 282

9 Multimodale Large Language Models . . . 283
Vision Transformer . . . 284
Multimodale Embedding-Modelle . . . 287
CLIP – Modelle, die eine Verbindung zwischen Texten und Bildern herstellen können . . . 289
Wie werden bei CLIP multimodale Embeddings generiert? . . . 289
OpenCLIP . . . 292
CLIP mit der sentence-transformers-Bibliothek laden . . . 296
Multimodale Textgenerierungsmodelle erstellen . . . 296
BLIP-2 – Modelle erstellen, die auf Basis von Texten und Bildern logische Schlüsse ziehen können . . . 297
Multimodale Eingabedaten aufbereiten . . . 301
1. Anwendungsfall: Bildbeschriftungen erstellen . . . 304
2. Anwendungsfall: Chatmodelle erstellen, die multimodale Prompts unterstützen . . . 306
Zusammenfassung . . . 309

Teil III: Sprachmodelle trainieren und feintunen . . . 311

10 Text-Embedding-Modelle erstellen . . . 313
Embedding-Modelle . . . 313
Was genau ist Contrastive Learning? . . . 316
SBERT . . . 318
Embedding-Modelle erstellen . . . 320
Kontrastive Beispiele erstellen . . . 321
Das Modell trainieren . . . 322
Differenziertere Evaluierung . . . 325
Verlustfunktionen . . . 326
Embedding-Modelle feintunen . . . 334
Embedding-Modelle mit gelabelten Datensätzen feintunen . . . 335
Augmented SBERT . . . 337
Embedding-Modelle mit ungelabelten Daten feintunen . . . 342
Transformer-based Sequential Denoising Auto-Encoder (TSDAE) . . . 342
TSDAE zur Domain Adaptation nutzen . . . 346
Zusammenfassung . . . 347

11 Representation-Modelle für die Klassifikation feintunen 349
Klassifikation mit gelabelten Daten 349
Ein vortrainiertes BERT-Modell feintunen 351
Schichten eines Modells einfrieren 354
Few-Shot-Klassifikation 359
SetFit – Modelle mit nur wenigen Trainingsbeispielen auf effiziente Weise feintunen 359
Modelle für die Few-Shot-Klassifikation feintunen 363
Bereits vortrainierte Modelle mittels Masked Language Modeling weiter vortrainieren 366
Named-Entity Recognition 371
Daten für die Named-Entity Recognition aufbereiten 373
Modelle für die Named-Entity Recognition feintunen 378
Zusammenfassung 379

12 Generative Modelle feintunen 381
Die drei Schritte beim Training von LLMs – Pretraining, Supervised Fine-Tuning und Preference Tuning 381
Supervised Fine-Tuning (SFT) 383
Vollständiges Feintuning (Full Fine-Tuning) 384
Parameter-efficient Fine-Tuning (PEFT) 385
Instruction Tuning mit dem QLoRA-Verfahren 393
Instruktionsdaten über ein Template bereitstellen 394
Modelle quantisieren 395
Die Konfiguration für das LoRA-Verfahren festlegen 396
Die Parameter für das Training festlegen 397
Das Modell trainieren 398
Gewichte des QLoRa-Modells mit denen des Basismodells zusammenführen 399
Generative Modelle evaluieren 400
Metriken auf Wortebene 400
Benchmarks 401
Leaderboards 402
Automatisierte Evaluierung 403
Evaluierung durch Menschen 403
Modelle darauf ausrichten, was Nutzer erwarten – Preference Tuning, Alignment und Reinforcement Learning from Human Feedback 405
Die Preference Evaluation mithilfe von Reward-Modellen automatisieren 407
Die Ein- und Ausgaben des Reward-Modells 407
Ein Reward-Modell trainieren 408
Preference Tuning ohne zusätzliches Training eines Reward-Modells 411

Preference Tuning mittels DPO 414
Templates für Präferenzdatensätze erstellen 414
Das Modell quantisieren 415
Parameter für das Training festlegen 416
Das Modell trainieren 416
Zusammenfassung 417

Schlussbemerkung **419**

Index **421**

Index

A

Accuracy *siehe* Treffergenauigkeit
Adaptive Pretraining 347
Agenten 240
  RAG-Systeme mit Agenten 280
  ReAct im LangChain-Framework 244
  Schlüsse aus mehreren aufeinanderfolgenden Gedankenschritten ziehen 242
ALBERT 135
align_labels-Funktion 376
all-MiniLM-L6-v2-Modell 335
all-mpnet-base-v2-Modell 363
Annoy 263
Anthropic Claude 53
APIs (Application Programming Interfaces) 53
  Cohere 19, 254
  Embeddings erstellen 143
  externe 154
  OpenAI 19, 153
ArXiv 158
Attention 35
  Erklärung 35
  Transformer-Architektur 38
    Attention-Mechanismus 112
    Attention-Schicht 103, 110–111, 128
    Berechnung von Attention 114
    Flash-Attention 122
    Grouped-Query-Attention 120
    Local-Attention 118
    Multi-Query-Attention 120
    Optimierung des Attention-Mechanismus 120
    Self-Attention und Relevanzbewertung 116
    Sparse-Attention 118
Attention-Heads 114, 128
aufgabenspezifische Modelle 133
Augmented SBERT 337
Ausreißer in Daten 164
autoregressive Architektur 36, 39, 46, 99, 120

B

Bag-of-Words-Modell 30
  Embeddings 34
  Topic Modeling 168, 176
Benchmarks, bei Evaluierung generativer Modelle 401
BERT (Bidirectional Encoder Representations from Transformers) 41
  Feintuning vortrainierter BERT-Modelle 351
  Masked Language Modeling 336
  Transformer-Blöcke im Vergleich zu 120
  Varianten des BERT-Modells 135
  Vergleich mit anderen trainierten Tokenizern 71
  Verwendung in Suchmaschinen 249
BERTopic 168
  modularer Aufbau 172
  Representation-Blöcke 176
    KeyBERTInspired 179
    Maximal Marginal Relevance 180
    Textgenerierung 181
  Varianten von Algorithmen 172
BERTScore 400
Beurteilung durch Menschen 165
Bidirectional Encoder Representations from Transformers *siehe* BERT

Bildbeschriftungen erstellen 304
Bilder *siehe* multimodal
bitsandbytes-Package 395
BLEU 400
BLIP-2 (Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation 2)
  Bildbeschriftungen erstellen 304
  Chatmodelle, die multimodale Prompts unterstützen 306
  Q-Former 297
  Texte aufbereiten 303
BM25-Algorithmus 257
BPE (Byte Pair Encoding) 67, 79

C

Chains 203, 225
  Chain-of-Thought 206
  mehrere Prompts miteinander verketten 228
  Prompt-Templates mit einem LLM verketten 225
chat_history (Eingabevariable) 233
Chatbot Arena 404
ChatGPT 224, 296, 411, 414
  Textklassifikation 152
  Veröffentlichung 27
chatgpt_generation-Funktion 153
Chatmodelle, die multimodale Prompts unterstützen 306
Chat-Token 77
Chunking von Texten 254, 259
  Ansätze 261
  ein Vektor pro Dokument 259
  mehrere Vektoren pro Dokument 260
CLIP 289
  multimodale Embeddings erstellen 289
  OpenCLIP 292
  Verbindung zwischen Texten und Bildern 289
Closed-Source-LLMs 53
Clustering *siehe* Text-Clustering
CNNs (Convolutional Neural Networks) 284
Cohere 276
  Command R+ 54, 281
  Embeddings erstellen 143
  Nutzerkonto erstellen 19, 254
  Reranking-API 265
  Suchanfragen umformulieren 279
Completion-Modelle 45
compute_metrics-Funktion 377
Computerlinguistik *siehe* Natural Language Processing (NLP)
CoNLL-2003-Datensatz 373
Contrastive Learning
  Text-Embedding-Modelle 316, 321
  Word2vec-Algorithmus und 87
Contrastive Tension (CT) 342
Conversation Buffer (Speicher für den Gesprächsverlauf) 240
  Erklärung 233
  Windowed Conversation Buffer 235
Conversation Summary (Speicher für den Gesprächsverlauf) 236
convert_ids_to_tokens-Funktion 293
Convolutional Neural Networks (CNNs) 284
CountVectorizer 171
Cross-Encoder 268
CT (Contrastive Tension) 342
c-TF-IDF 170, 179, 181

D

DataCollator-Klasse 352, 368
datamapplot-Paket 184
DBSCAN (Density-based Spatial Clustering) 164
DeBERTa 83, 135
Decodierungsstrategie 103, 127
Denkweise nach zwei verschiedenen Systemen 206
Dense Retrieval 250, 252
  Beispiel 254
  Chunking von Texten 259
  Embedding-Modelle feintunen für 263
  mögliche Probleme 258
  Suche auf Basis der nächsten Nachbarn mithilfe von Vektordatenbanken skalieren 262
Density-based Spatial Clustering (DBSCAN) 164
dichtebasierter Algorithmus 163
DistilBERT 135, 140
do_sample-Parameter 58
Domain Adaptation 346
DPO (Direct Preference Optimization) 411
  Feintuning 416
  Modelle quantisieren 415

Parameter für das Training festlegen 416
Templates für Präferenzdatensätze erstellen 414
DSPy 222

E

Easy Negatives (einfache Negativbeispiele) 333
Einbettungen *siehe* Embeddings
Einfrieren von Schichten 323, 354
eingefrorene (nicht trainierbare) Modelle 134, 141, 349
Embeddings 32, 61, 80
Arten von 34
Dense Retrieval 250, 252
Embedding-Modelle, Erklärung 134, 314
Empfehlungssysteme 91
Erklärung 32, 313
multimodal 287
CLIP 289
OpenCLIP 292
Positional-Embeddings 124
Text-Embedding-Modelle 85, 313
Contrastive Learning 316
erstellen 320
Feintuning 334
SBERT 318
Unsupervised Learning 342
Texte mit Embedding-Modellen klassifizieren 140
überwachte Klassifikation 141
Zero-Shot-Klassifikation 143
Token-Embeddings 80, 85
kontextualisierte Word-Embeddings erstellen 82
Vokabular des Tokenizers und 81
Vorgehensweise beim Text-Clustering 159
Cluster inspizieren 164
Embedding-Modell 159
Modell für Clustering 163
Modell zur Dimensionsreduktion 160
Word-Embeddings 86
vortrainierte 86
Word2vec-Algorithmus und Training mittels Contrastive Learning 87
Empfehlungssysteme 91
Empfehlungssysteme für Songs 91
Encoder-Decoder-Modelle 148
engere Auswahl treffen 266
Evaluierung durch Menschen 403
Exponential Backoff 154

F

F1-Maß, Wahrheitsmatrix 139
F1-Score *siehe* F1-Maß
FAISS 263
Falcon 188
Feature-Extraction-Schritt bei Klassifikation mit Embedding-Modell 141
Feedforward Neural Networks 39, 110, 128, 352
Feedforward-Schicht 110
Feintuning
Embedding-Modelle für Dense Retrieval 263
Erklärung 49
generative Modelle 214, 381
evaluieren 400
Preference Tuning 383, 405
Supervised Fine-Tuning 382
Trainingsschritte 381
Representation-Modelle 349
Few-Shot-Klassifikation 359
Masked Language Modeling 366
Named-Entity Recognition 371
überwachte Klassifikation 349
T5-Modell 149
Text-Embedding-Modelle 334
Augmented SBERT 337
überwacht (Supervised) 335
Few-Shot-Klassifikation 359
Feintuning für Klassifikation 363
SetFit 359
Few-Shot-Prompting 201, 214
find_topics()-Funktion, BERTopic 174
Flan-T5-Modell 74, 150
Flash-Attention 122, 128
Formatierung 216
Forward Pass 127
Erklärung 98
Komponenten, die durchlaufen werden 100
Foundation-Modelle 47
fp16-Parameter 324

G

Galactica 76
gefährliche Inhalte erstellen 52

geistiges Eigentum 52
General-Language-Understanding-Evaluation-(GLUE-)Benchmark 321, 325, 386, 401
generated_text-Variable 305–306
generation_output-Variable 66
generative Modelle 44
  evaluieren 400
    automatisierte Evaluierung 403
    Benchmarks 401
    Evaluierung durch Menschen 403
    Leaderboards 402
    Metriken auf Wortebene 400
  Feintuning 381
    Evaluierung 400
    Preference Tuning 383, 405
    Supervised Fine-Tuning 382
    Trainingsschritte 381
  Reasoning 205
    Chain-of-Thought 206
    Self-Consistency 209
    Tree-of-Thought 210
  Representation-Modelle im Vergleich zu 43
  Textklassifikation 147
    ChatGPT 152
    T5 148
Generative Pre-trained Transformers *siehe* GPTs
Generative Pseudo-Labeling (GPL) 342
Gensim-Bibliothek 86
get_topic_info()-Methode 173
get_topic-Funktion 174
GGUF-Modellformat 217, 222
GitHub 21
GloVe 318
GLUE-(General-Language-Understanding-Evaluation-)Benchmark 321, 325, 386, 401
Goldstandard-Datensätze 337
Goodharts Gesetz 404
Google Colab 16, 18, 53, 57, 108, 399
Google Gemini 274
Google Search 249
GPL (Generative Pseudo-Labeling) 342
GPT2Tokenizer 303
GPTs
GPTs (Generative Pre-trained Transformers) 44, 187
  GPT-1 44
  GPT-2 27, 45, 69, 73
  GPT-3 45, 110, 118, 390
  GPT-3.5 46, 152, 228, 244
  GPT-4 46, 53, 62, 74, 110
  siehe auch Textgenerierung
GPUs
  Flash-Attention 122
  SRAM und HBM 122
  Voraussetzungen 18, 52
GPUs von NVIDIA 18, 57
Grafikspeicher *siehe* VRAM (Video Random-Access Memory)
Greedy Decoding 104
Großschreibung 79
Grounded Generation 275
Grouped-Query-Attention 120, 128
GSM8k 401
Ġ-Symbol 303
Guardrails 216
Guidance 216

H

Halluzinationen
  Textgenerierungsmodelle 249
  vermeiden beim Instruction-based Prompting 196
Hard Negatives (schwere Negativbeispiele) 333
Haystack 222
HBM (High Bandwidth Memory) 122
HDBSCAN (Hierarchical Density-based Spatial Clustering of Applications with Noise) 164, 174
HellaSwag 401
High Bandwidth Memory (HBM) 122
Hugging Face 56, 132
  evaluate-Paket 377
  Nutzerkonto erstellen 19
  Tokenizer 80
HumanEval 402
hybride Suche 258, 266

I

Idefics 2 300
In-Context Learning 201
Indexe 255
InfoNCE 330
input_ids-Variable 64
Instruction-based Prompting 196
invoke-Funktion 227

J

Jahr der generativen KI 46
Jupyter 108

K

KeyBERTInspired 179, 181
KI (künstliche Intelligenz)
  Erklärung 28
  Fortschritte bei Entwicklung 27
Klassifikationsberichte 138
Klassifikationsschritt bei Klassifikation mit Embedding-Modell 141
k-Means-Algorithmus 163, 172, 174
Kontext
  Attention und 111
  Prompt Engineering 198
  Trainingsdatensätze 87
Kontextfenster bei generativen Modellen 46
Kontextlänge
  Completion-Modelle 46
  maximal mögliche Anzahl zu verarbeitender Tokens 106
Kosinus-Ähnlichkeit 145, 327, 329
Kreuzentropie-Verlustfunktion 331
künstliche Intelligenz *siehe* KI
KV-(Keys-and-Values-)Cache 107

L

LangChain 221
  quantisierte Modelle laden mit 222
  ReAct in 244
  siehe auch Chains
Language AI (Language Artificial Intelligence) 27
  Erklärung 28
  jüngste Entwicklungen 29
    Attention 35
    Bag-of-Words-Modell 30
    Embeddings 32
    generative Modelle 44
    Jahr der generativen KI 46
    Representation-Modelle 41
Language Modeling 381
Language-Modeling-Head (LM-Head) 100
Large Language Models *siehe* LLMs
Latent Dirichlet Allocation 167
LayerNorm 124
Leaderboards, bei Evaluierung generativer Modelle 402
learning_rate-Parameter 398
Leerzeichen 73
leicht verständlicher Zugang 15
Llama 188
Llama 2 19, 52, 77, 120, 188, 296
llama-cpp-python-Bibliothek 217
Llama-Modell von Meta 54
LLaVA 300
LLMs (Large Language Models) 27
  Ansatz beim Training 48
  Anwendungsmöglichkeiten 50
  Codebeispiele und Übungen 21
  Embeddings 61, 80
    Empfehlungssysteme 91
    Text-Embeddings 85
    Token-Embeddings 80
    Word-Embeddings 86
  Feintuning von generativen Modellen 381
    Evaluierung 400
    Preference Tuning 383, 405
    Supervised Fine-Tuning 382
    Trainingsschritte 381
  Feintuning von Representation-Modellen 349
    Few-Shot-Klassifikation 359
    Masked Language Modeling 366
    Named-Entity Recognition 371
    überwachte Klassifikation 349
  generative Modelle 44
  Hardware- und Softwarevoraussetzungen 18, 52
  Interaktion mit 53
    Closed-Source-Modelle 53
    Open-Source-Modelle 54
  jüngste Entwicklungen im Bereich der Language AI 29
  leicht verständlicher Zugang 15
  multimodal 283
    Embedding-Modelle 287
    Textgenerierungsmodelle 296
    Vision Transformer 284
  Prompt Engineering 187
    Elemente eines Prompts 194
    In-Context Learning 201
    Instruction-based Prompting 196
    Modellausgaben validieren 213
    Prompt Chaining 203
    Prompts komplexer gestalten 198
    Reasoning mit generativen Modellen 205
    Textgenerierungsmodelle 187

Representation-Modelle 41
Retrieval-Augmented Generation 251, 273
  Antworten evaluieren 281
  Grounded Generation 276
  mehrere Suchanfragen auf einmal stellen 279
  mit lokalen Modellen 276
  RAG-Systeme mit Agenten 280
  Suchanfragen umformulieren 279
  Suchsysteme zu RAG-Systemen erweitern 274
  unterschiedliche Datenquellen für verschiedene Suchanfragen vorsehen 280
semantische Suche 249
  Dense Retrieval 250, 252
  Metriken zur Evaluierung von Retrieval-Systemen 268
  Reranking 250, 264
sich wandelnde Definition von 48
Text-Clustering 157
Text-Embedding-Modelle 313
  Contrastive Learning 316
  erstellen 320
  Feintuning 334
  SBERT 318
  Unsupervised Learning 342
Textgenerierung 56, 221
  Agenten 240
  Chains 225
  Optimierungen hinsichtlich der Verwendung von Modellen 222
  Speicher für den Gesprächsverlauf 231
Textklassifikation 131
  mit generativen Modellen 147
  mit Representation-Modellen 133
  Spielfilmrezensionen 132
Tokens und Tokenizer 61
  Eingaben aufbereiten 62
  entscheidende Faktoren bei Tokenisierung 67
  LLMs herunterladen und ausführen 63
  Token-Embeddings 80
  Tokenisierungsverfahren 68
  Vergleich verschiedener trainierter Tokenizer 70
  Verhalten von Tokenizern 78
Topic Modeling 158, 167
Transformer-Architektur 97
  Decodierungsstrategie 103
  Ein- und Ausgaben eines Transformer-basierten LLM 98
  Keys-and-Values-Cache 107
  Komponenten, die beim Forward-Pass durchlaufen werden 100
  parallele Verarbeitung von Tokens und Kontextlänge 105
  Transformer-Blöcke 109
  Verbesserungen in jüngster Zeit 118
verantwortungsvoll entwickeln und nutzen 51
von außen betrachtet 62
LM-Head (Language-Modeling-Head) 100
LMQL 216
Local-Attention 118
lora_alpha-Parameter 397
LoRA (Low-Rank Adaptation) 388, 415
  siehe auch QLoRA
lr_scheduler_type-Parameter 398

M

Mamba 47
MAP (Mean Average Precision) 268
MarginMSE-Verlustfunktion 327
mask_token 71
Masked Language Modeling (MLM) 366
Massive Text Embedding Benchmark (MTEB) 136, 160, 277, 325
matplotlib-Bibliothek 166
max_new_tokens-Parameter 58
Maximal Marginal Relevance (MMR) 180
McCarthy, John 28
mehrere Prompts gleichzeitig verwenden 205
mehrere Suchanfragen auf einmal stellen 279
Metriken auf Wortebene, bei Evaluierung generativer Modelle 400
Metriken zur Evaluierung von Retrieval-Systemen 268
  Bewertung im Fall einer einzigen Suchanfrage 271
  Bewertung im Fall mehrerer Suchanfragen 272
microsoft/mpnet-base-Modell 322
Microsoft Bing 249, 265
Microsoft Bing AI 274
min_cluster_size-Parameter 164
min_dist-Parameter 162
MIRACL 267

Mistral 54, 188, 300
MLM (Masked Language Modeling) 366
MMLU 401, 403
MMR (Maximal Marginal Relevance) 180
MNLI-(Multi-Genre-Natural-Language-Inference-)Korpus 321, 331
MNR (Multiple Negatives Ranking) Loss 330
Modellausgaben beschränken 214, 216
Modellausgaben validieren 213
  Beispiele bereitstellen 214
  ethische Aspekte 213
  faktische Korrektheit 214
  Formatierung der Ausgabe 213
  Gültigkeit 213
  Modellausgaben beschränken 216
Modell für Clustering 163
Modell zur Dimensionsreduktion 160
monoBERT 268
MTEB (Massive Text Embedding Benchmark) 136, 160, 277, 325
Multi-Genre-Natural-Language-Inference-(MNLI-)Korpus 321, 331
Multi-Layer Perceptron 103
  siehe auch Feedforward Neural Networks
multimodal 283
  Embedding-Modelle 287
    CLIP 289
    OpenCLIP 292
  Erklärung 283
  Textgenerierungsmodelle 296
    Bildbeschriftungen erstellen 304
    Bilder aufbereiten 302
    BLIP-2 297
    Chatmodelle, die multimodale Prompts unterstützen 306
    Texte aufbereiten 303
  Vision Transformer 284
Multiple Negatives Ranking (MNR) Loss 330
Multi-Query-Attention 120

N

n_components-Parameter 162
Named-Entity Recognition *siehe* NER
Natural Language Inference (NLI) 321
Natural Language Processing (NLP) 28, 317
nDCG (Normalized Discounted Cumulative Gain) 267, 273
Negative Sampling 89
NER (Named-Entity Recognition) 371, 386
  Daten aufbereiten für 373
  Feintuning für 378
neuronale Netze 32
neuronales Feedforward-Netz *siehe* Feedforward Neural Networks
Nicht-Spieler-Charakter 28
nicht trainierbare (eingefrorene) Modelle 134, 141, 349
NLI (Natural Language Inference) 321
NLP (Natural Language Processing) 28, 317
Noise-contrastive Estimation 89
Normalisierung, Transformer-Block 123
Normalized Discounted Cumulative Gain (nDCG) 267, 273
NTXentLoss 330
Nucleus Sampling 192
num_train_epochs-Parameter 324, 398
NumPy 262

O

Odds Ratio Preference Optimization (ORPO) 417
One-Shot-Prompting 203
  Chain-of-Thought im Vergleich zu 207
  In-Context Learning 201
OpenAI 152
  Embeddings erstellen 143
  Nutzerkonto erstellen 19, 153
  siehe auch ChatGPT
OpenCLIP 292
Open LLM Leaderboard 224, 403
Open-Source-LLMs 54
Optimierungen hinsichtlich der Verwendung von Modellen 222
optim-Parameter 398
ORPO (Odds Ratio Preference Optimization) 417

P

pad_token 71
parallele Verarbeitung 114, 128
PCA (Principal Component Analysis) 162
peft_config-Parameter 397
PEFT (Parameter-efficient Fine-Tuning) 385
  Adapter 385
  Komprimierung 390
  LoRA 388
peft-Bibliothek 396
per_device_eval_batch_size-Argument 324

per_device_train_batch_size-Argument 324
Perplexity 274
Persona, bei Prompts, mit denen ein Text generiert werden soll 198
Phi-3
  Forward Pass 103
  Prompt-Template 226
  quantisierte Modelle laden 224
  Quantisierung 223
  Vergleich mit anderen trainierten Tokenizern 77
Phi-3-mini 56, 188
Phi-Modell von Microsoft 54
Pinecone 263
Positional-Embeddings 124
PPO (Proximal Policy Optimization) 411
Precision *siehe* Relevanz
Preference Tuning 152, 383, 405
  Direct Preference Optimization 411
    Feintuning 416
    Modelle quantisieren 415
    Parameter für das Training festlegen 416
    Templates für Präferenzdatensätze erstellen 414
  Reward-Modelle 407
    Ein- und Ausgaben von 407
    Training 408
Pretraining, Erklärung 49
Primacy-Effekt 197
Principal Component Analysis (PCA) 162
Projektionsmatrizen 115
Prompt Engineering 147, 187
  Elemente eines Prompts 194
  In-Context Learning 201
  Instruction-based Prompting 196
  Modellausgaben validieren 213
    Beispiele bereitstellen 214
    Modellausgaben beschränken 216
  Prompt Chaining 203
  Prompts komplexer gestalten 198
  Reasoning mit generativen Modellen 205
  Textgenerierungsmodelle 187
    laden 188
    Modellausgabe beeinflussen 190
    wählen 187
Proximal Policy Optimization (PPO) 411
Python lernen 16

Q

Q-Former (Querying Transformer) 297
QLoRA (Quantized Low-Rank Adaptation) 393
  Feintuning 398
  Gewichte zusammenführen 399
  Instruktionsdaten mit einem Template bereitstellen 394
  Modelle quantisieren 395
  Parameter für das Training festlegen 397
  Parameter für LoRA festlegen 396
quantisiertes Modell 224
Quantisierung 223, 390
quantization_config-Parameter 397
Quantized Low-Rank Adaptation *siehe* QLoRA
Querying Transformer (Q-Former) 297

R

r (Parameter) 397
RAG (Retrieval-Augmented Generation) 81, 86, 251, 273
  Antworten evaluieren 281
  Grounded Generation 276
  mehrere Suchanfragen auf einmal stellen 279
  mit lokalen Modellen 276
  RAG-Systeme mit Agenten 280
  Suchanfragen umformulieren 279
  Suchsysteme zu RAG-Systemen erweitern 274
  typische Pipeline 274
  unterschiedliche Datenquellen für verschiedene Suchanfragen vorsehen 280
Ragas 281
random_state-Parameter 162
Rate Limit Errors 154
ReAct
  im LangChain-Framework 244
  Schlüsse aus mehreren aufeinanderfolgenden Gedankenschritten ziehen 242
Reasoning
  mehrere aufeinanderfolgende Gedankenschritte 242, 244
  mit generativen Modellen 205
    Chain-of-Thought 206
    Self-Consistency 209
    Tree-of-Thought 210

Recall *siehe* Trefferquote
Recency-Effekt 197
Recurrent Neural Networks (RNNs) 35
reduce_outliers()-Funktion 174
Regulierung 52
rein Decoder-basierte Modelle (Decoder-Only) *siehe* generative Modelle
rein Encoder-basierte Modelle (Encoder-Only) *siehe* Representation-Modelle
Relevanz, Wahrheitsmatrix 139
Relevanzbewertung 113, 115, 125
Repository 19
representation_model-Parameter 182
Representation-Modelle 41
  aufgabenspezifische Modelle 140
  Erklärung 31
  Feintuning für Klassifikation 349
    Few-Shot-Klassifikation 359
    Masked Language Modeling 366
    Named-Entity Recognition 371
    überwachte Klassifikation 349
  generative Modelle im Vergleich zu 43
  Textklassifikation 133
    aufgabenspezifische Modelle 136
    Modell auswählen 135
    Texte mit Embedding-Modellen klassifizieren 140
Reranking 250, 264
  Beispiel 265
  BERTopic 177
  Funktionsweise von Reranking-Modellen 268
  Sentence-Transformers-Bibliothek 267
Retrieval-Augmented Generation *siehe* RAG
return_full_text parameter 58
Reward-Modelle 407
  Ein- und Ausgaben von 407
  Training 408
RMSNorm 124
RNNs (Recurrent Neural Networks) 35
RoBERTa 69, 135
RoPE (Rotary-Positional-Embeddings) 124
Rorschach-Test 305
Rotten-Tomatoes-Datensatz 132, 351
ROUGE 400
RWKV 47

S

SBERT 318, 337
Schlagwortsuche
  mit Ergebnissen für semantische Suche vergleichen 257
  Reranking 266
Schlüsse aus mehreren aufeinanderfolgenden Gedankenschritten ziehen 242
Schreibstil, bei Prompts, mit denen ein Text generiert werden soll 199
Self-Attention 39, 116
Self-Consistency 209
Semantic Textual Similarity Benchmark (STSB) 323
semantische Suche 249
  Bedeutung 249
  Dense Retrieval 250, 252
    Beispiel 254
    Chunking von Texten 259
    Embedding-Modelle feintunen für 263
    mögliche Probleme 258
    Suche auf Basis der nächsten Nachbarn mithilfe von Vektordatenbanken skalieren 262
  Metriken zur Evaluierung von Retrieval-Systemen 268
    Bewertung im Fall einer einzigen Suchanfrage 271
    Bewertung im Fall mehrerer Suchanfragen 272
  Reranking 250, 264
    Beispiel 265
    Funktionsweise von Reranking-Modellen 268
    Sentence-Transformers-Bibliothek 267
Semi-Hard Negatives (mittelschwere Negativbeispiele) 334
SentencePiece 74
Sentence-Transformers-Bibliothek 85, 141, 172, 267, 296, 318, 335, 359
Sequence-to-Sequence-Modelle 147–148
SetFit 349, 359
SFT (Supervised Fine-Tuning) 382
  Parameter-efficient Fine-Tuning (PEFT) 385
    Adapter 385
    Komprimierung 390
    LoRA 388
  QLoRA 393
    Feintuning 398
    Gewichte zusammenführen 399
    Instruktionsdaten mit einem Template bereitstellen 394

Modelle quantisieren 395
Parameter für das Training festlegen 397
Parameter für LoRA festlegen 396
vollständiges Feintuning 384
Shared Memory (SRAM) 122
Silberstandard-Datensätze 338
SimCSE (Simple Contrastive Learning of Sentence Embeddings) 342
Skip-Gram 89
Softmax-Verlustfunktion 326
Sondertokens 70, 79
Sparse-Attention 118
Special Tokens *siehe* Sondertokens
Speicher für den Gesprächsverlauf 231
Conversation Buffer 233
Conversation Summary 236
Windowed Conversation Buffer 235
Spezifität beim Instruction-based Prompting 196
SRAM (Shared Memory) 122
StableLM 188
StarCoder2 75
STSB (Semantic Textual Similarity Benchmark) 323
Suche auf Basis der nächsten Nachbarn
Empfehlungssysteme, die auf Embeddings basieren 92
Vektordatenbanken 262
vortrainierte Word-Embeddings 87
Supervised Fine-Tuning *siehe* SFT

T

T5 (Text-to-Text Transfer Transformer) 148
target_modules-Parameter 397
temperature-Parameter 191, 209
Tesla T4 399
Testdatensätze 133, 137
Text-Clustering 157
bewährte Vorgehensweise beim 159
Cluster inspizieren 164
Embedding-Modell 159
Modell für Clustering 163
Modell zur Dimensionsreduktion 160
CLIP (Embedding-Modell) und 289
Text-Embedding-Modelle 85, 313
Contrastive Learning 316
erstellen 320
evaluieren 325
kontrastive Beispiele erstellen 321
Training 322
Verlustfunktionen 326
Feintuning 334
Augmented SBERT 337
überwacht (Supervised) 335
SBERT 318
Unsupervised Learning 342
Textgenerierung 56, 221
Agenten 240
ReAct im LangChain-Framework 244
Schlüsse aus mehreren aufeinanderfolgenden Gedankenschritten ziehen 242
Chains 225
mehrere Prompts miteinander verketten 228
Prompt-Templates mit einem LLM verketten 225
multimodal 296
Bildbeschriftungen erstellen 304
Bilder aufbereiten 302
BLIP-2 297
Chatmodelle, die multimodale Prompts unterstützen 306
Texte aufbereiten 303
Optimierungen hinsichtlich der Verwendung von Modellen 222
Prompt Engineering 187
geeignetes Modell wählen 187
Modellausgabe beeinflussen 190
Modelle laden 188
Speicher für den Gesprächsverlauf 231
Conversation Buffer 233
Conversation Summary 236
Windowed Conversation Buffer 235
Topic Modeling 181
Text-in-Text-out-Modell 98
Textklassifikation 131
mit generativen Modellen 147
ChatGPT 152
T5 148
mit Representation-Modellen 133
aufgabenspezifische Modelle 136
Modell auswählen 135
Texte mit Embedding-Modellen klassifizieren 140
Spielfilmrezensionen 132
Text-to-Text Transfer Transformer (T5) 148, 152
thenlper/gte-small-Modell 160
%%timeit (Magic Command) 108

TinyLlama 393, 414
[CLS]-Token 42, 71, 84, 294, 318, 344, 377
[SEP]-Token 71, 84, 377
<work>-Token 77
Tokenisierung auf Byte-Ebene 69
Tokenisierung auf Wortebene 68
Tokenisierung auf Wortteilebene 68
Tokenisierung auf Zeichenebene 68
Tokenization-free Encoding 69
Tokens und Tokenizer 57, 61
  aufgabenspezifisches Representation-Modell 136, 147
  Ausrichtung auf Texte im Vergleich zu Code 80
  Bag-of-Words-Modell 30
  Decodierungsstrategie 103
  Eingaben aufbereiten 62
  entscheidende Faktoren bei Tokenisierung 67
  Forward Pass 100
  Leerzeichen 73
  LLMs herunterladen und ausführen 63
  Masked Language Modeling 149
  parallele Verarbeitung von Tokens und Kontextlänge 105
  Sondertokens 70
  Token-Embeddings 80, 101, 127
    kontextualisierte Word-Embeddings erstellen 82
    Vokabular des Tokenizers und 81
  Tokenisierungsverfahren 68
    Tokenisierung auf Byte-Ebene 69
    Tokenisierung auf Wortebene 68
    Tokenisierung auf Wortteilebene 68
    Tokenisierung auf Zeichenebene 68
  Token Spans 149
  Vergleich verschiedener trainierter Tokenizer 70
    BERT-Base-Modell (cased) 72
    BERT-Base-Modell (uncased) 71
    Flan-T5 74
    Galactica 76
    GPT-2 73
    GPT-4 74
    Phi-3- und Llama-2-Modell 77
    StarCoder2 75
  Verhalten von Tokenizern 78
    Datensätze 80
    Parameter 79
    Tokenisierungsverfahren 79
top_k-Parameter 192
top_p-Parameter 192, 209
Topic Modeling 158, 167
  BERTopic 168
  Representation-Blöcke 176
TrainingArguments-Klasse 353
Trainingsdatensatz 133
Transfer Learning 43
Transformer-Architektur 38, 97
  Attention-Schicht 103, 110
  Decodierungsstrategie 103
  Ein- und Ausgaben eines Transformer-basierten LLM 98
  Feedforward-Schicht 110
  Keys-and-Values-Cache 107
  Komponenten, die beim Forward-Pass durchlaufen werden 100
  Optimierung des Attention-Mechanismus 120
  parallele Verarbeitung von Tokens und Kontextlänge 105
  Transformer-Blöcke 109
    Attention-Mechanismus 112
    Attention-Schicht 111
    Berechnung von Attention 114
    Feedforward Neural Networks 110
    Self-Attention und Relevanzbewertung 116
  Verbesserungen in jüngster Zeit 118
    effizienterer Attention-Mechanismus 118
    Positional-Embeddings 124
    Transformer-Blöcke 122
  Vision Transformer 284
Transparenz und Verantwortlichkeit 51
Tree-of-Thought 210
Treffergenauigkeit, Wahrheitsmatrix 139
Trefferquote, Wahrheitsmatrix 139
TruthfulQA 401, 403
TSDAE (Transformer-based Sequential Denoising Auto-Encoder)
  Erklärung 342
  für Domain Adaptation 346

U

überwachte Klassifikation 141
  Feintuning von Representation-Modellen für 349
    Einfrieren von Schichten 354
    vortrainierte BERT-Modelle 351
UltraChat-Datensatz 394

UMAP (Uniform Manifold Approximation and Projection) 162
Unigram-Sprachmodell 74
unk_token 71
use_cache-Parameter 108

V

Validierungsdatensatz 133
Validierung von Antworten, beim Prompt Chaining 205
Vektordatenbanken
  Dense Retrieval 253
  Retrieval-Augmented Generation 277
  Suche auf Basis der nächsten Nachbarn mithilfe von Vektordatenbanken skalieren 262
Verlustfunktionen 326
  auf Basis der Kosinus-Ähnlichkeit 329
  Multiple Negatives Ranking Loss 330
  Verlustfunktion auf Basis der Kosinus-Ähnlichkeit 327
Video Random-Access Memory (VRAM) 18, 52
Visualisierung
  BERTopic 176
  Clusteranalyse 165
  Dimensionsreduktion und 165
ViT (Vision Transformer) 284, 299
Vokabular von Tokenizern 79, 81, 101, 127
Vorhersagen treffen mit aufgabenspezifischen Modellen 138
Vorurteile und Fairness 51
VRAM (Video Random-Access Memory) 18, 52

W

Wahrheitsmatrix 138
warmup_ratio-Parameter 416
warmup_steps-Argument 324
Weaviate 263
Windowed Conversation Buffer (Speicher für den Gesprächsverlauf) 235, 240
Word2Vec-Algorithmus 32, 34
  Contrastive Learning und 87, 317
  Embeddings für Songs 91
Word-Embeddings 86
  vortrainierte 86
  Word2vec-Algorithmus und Training mittels Contrastive Learning 87
WordPiece 67
  BERT-Base-Modell (cased) 72
  BERT-Base-Modell (uncased) 71

Z

Zentroid-basierter Algorithmus 163
Zero-Shot-Klassifikation 143
  CLIP 289
  SetFit 365
Zero-Shot-Prompting
  Chain-of-Thought 208
  In-Context Learning 201
Zielgruppe, bei Prompts, mit denen ein Text generiert werden soll 199

O'REILLY®

Praxisbuch Large Language Models

Diese umfassende und anschauliche Einführung in die Welt der LLMs beschreibt sowohl konzeptionelle Grundlagen als auch konkrete Anwendungen und nützliche Tools. Tauchen Sie in das Innenleben von LLMs ein und erkunden Sie ihre Architekturen, Einsatzbereiche, Trainingsmethoden und Feintuning-Techniken. Mit seiner einzigartigen Mischung aus intuitiv verständlichen Illustrationen und praxisbezogenen Erläuterungen ist dieses Buch die ideale Ausgangsbasis für alle, die die Möglichkeiten von KI-Systemen voll ausschöpfen möchten.

Sie lernen, vortrainierte Transformer-LLMs von Hugging Face für Anwendungsfälle wie das Verfassen von Texten oder für Inhaltszusammenfassungen einzusetzen. Sie erfahren außerdem, wie Sie Suchsysteme erstellen und vorhandene Bibliotheken und vortrainierte Modelle für Textklassifikation, Suche und Clustering nutzen.

- **Verstehen Sie die Architektur von Transformer-basierten Sprachmodellen, die bei der Textgenerierung und -repräsentation hervorragende Ergebnisse liefern**
- **Entwerfen Sie fortgeschrittene LLM-Pipelines, um Textdokumente zu clustern und die darin enthaltenen Themen zu erforschen**
- **Erstellen Sie semantische Suchmaschinen, die über den Abgleich von Schlagwörtern hinausgehen und auf Methoden wie Dense Retrieval und Reranking basieren**
- **Lernen Sie, wie Sie generative Modelle optimal einsetzen – vom Prompt Engineering bis hin zur Retrieval Augmented Generation (RAG)**
- **Entwickeln Sie ein tieferes Verständnis dafür, wie LLMs trainiert und für spezifische Anwendungen optimiert werden, beispielsweise durch Feintuning generativer Modelle, Contrastive Fine-Tuning und In-Context-Learning**

»Jay und Maarten setzen ihre bewährte Arbeit fort, komplexe Themen mit hervorragenden Illustrationen und aufschlussreichen Beschreibungen zu erläutern. Für alle, die die wichtigsten Techniken zur Entwicklung von LLMs verstehen wollen, ist dieses Buch eine wertvolle Grundlage.«

— Andrew Ng
Gründer von DeepLearning.AI

Jay Alammar ist Director und Engineering Fellow bei Cohere.

Maarten Grootendorst ist Senior Clinical Data Scientist bei der Netherlands Comprehensive Cancer Organisation (IKNL).

9 783960 092667

www.dpunkt.de

Euro 49,90 (D)
ISBN 978-3-96009-266-7

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier