

## Praxisbuch Large Language Models

Sprache mit KI verarbeiten und generieren

» Hier geht's  
direkt  
zum Buch

# DIE LESEPROBE

# 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.

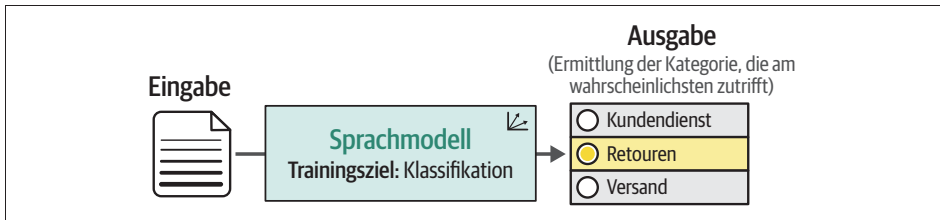


Abbildung 4.1: Sprachmodelle können auch zur Klassifizierung von Texten verwendet werden.

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).

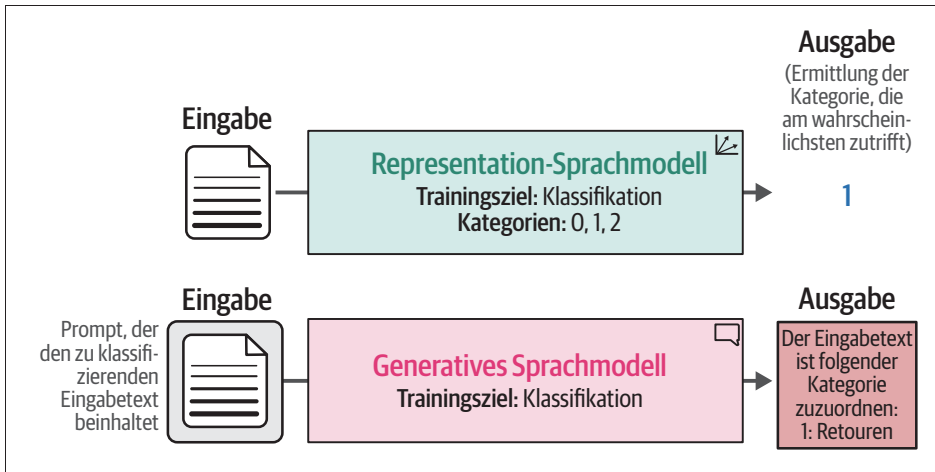


Abbildung 4.2: Sowohl Representation- als auch generative Modelle eignen sich für die Klassifikation. Allerdings unterscheiden sich beide Ansätze voneinander.

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.<sup>1</sup> Dieser enthält 5.331 positive und 5.331 negative Filmrezensionen von der Website *Rotten Tomatoes*.

<sup>1</sup> 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 .'],
 '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).

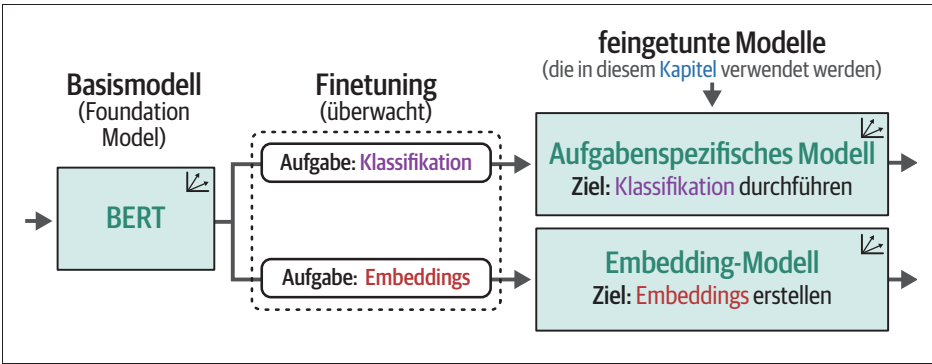


Abbildung 4.3: Ein Foundation-Modell wird für bestimmte Aufgaben feingetunt, beispielsweise für die Klassifikation oder die Generierung von Embeddings, die für verschiedenste Zwecke genutzt werden können.

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).

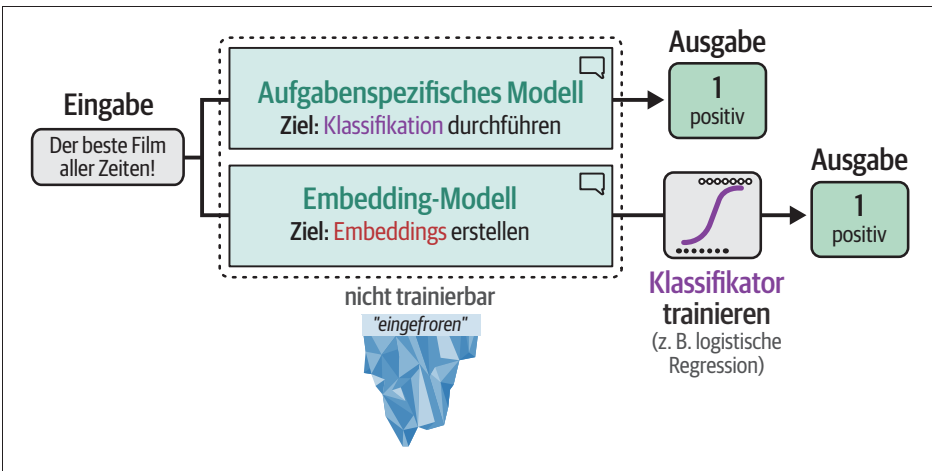


Abbildung 4.4: Eine Klassifikation kann entweder direkt mit einem aufgabenspezifischen Modell oder indirekt mit universell einsetzbaren Embeddings vorgenommen werden.

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<sup>2</sup>, DistilBERT<sup>3</sup>, ALBERT<sup>4</sup> und DeBERTa<sup>5</sup>. 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.

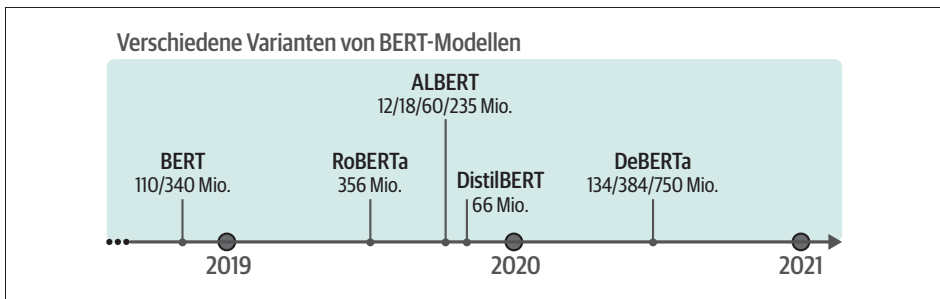


Abbildung 4.5: Eine chronologische Übersicht über bekannte Varianten des BERT-Modells. Hierbei handelt es sich um Basis- bzw. Foundation-Modelle, die hauptsächlich dafür gedacht sind, für eine nachgelagerte Aufgabe feingetunt zu werden.

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](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.

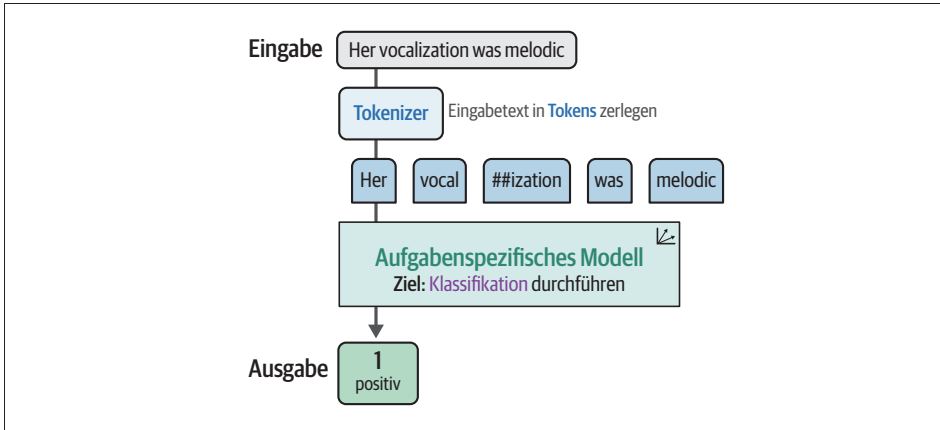


Abbildung 4.6: Bevor ein eingegebener Satz vom aufgabenspezifischen Modell verarbeitet werden kann, wird er zunächst an den Tokenizer geleitet.

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).

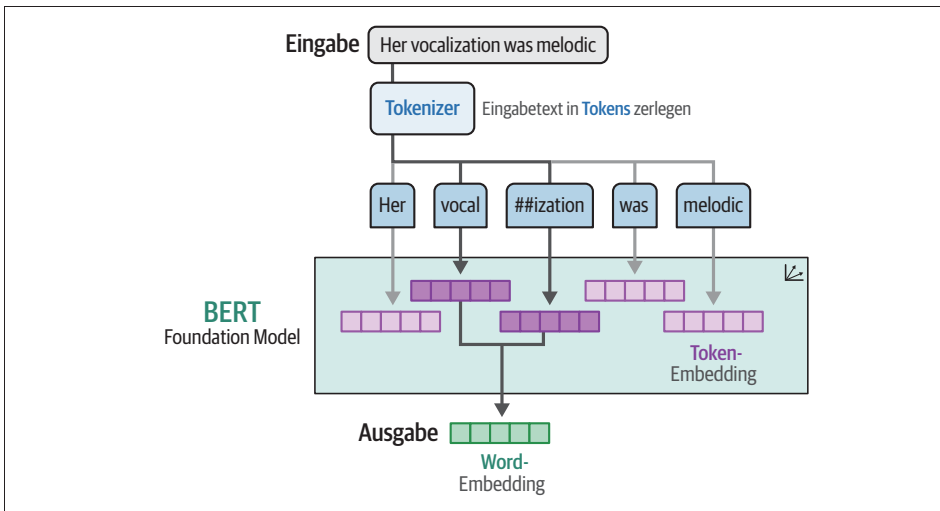


Abbildung 4.7: Selbst wenn ein Wort unbekannt ist bzw. nicht in den Trainingsdaten enthalten war, können in der Regel dennoch entsprechende Word-Embeddings generiert werden, da das Wort zunächst in Tokens zerlegt wird.

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).

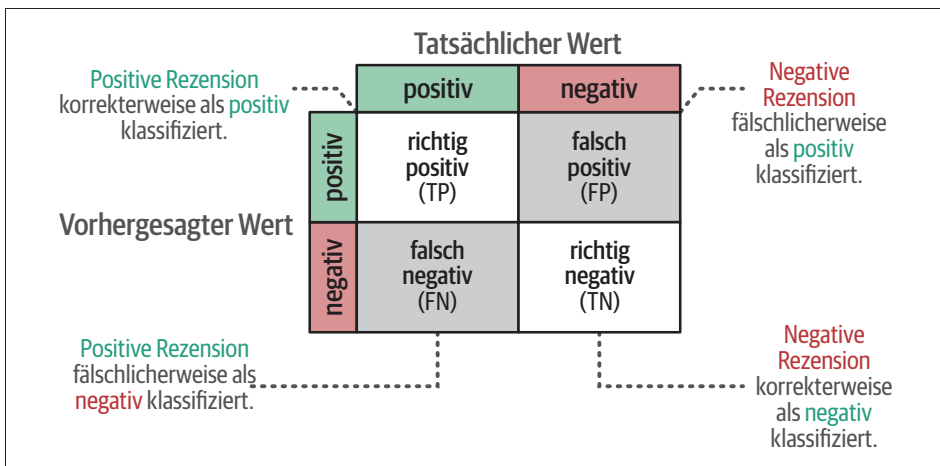


Abbildung 4.8: Die vier verschiedenen Arten von Vorhersagen lassen sich aus der Wahrheitsmatrix ablesen.

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.

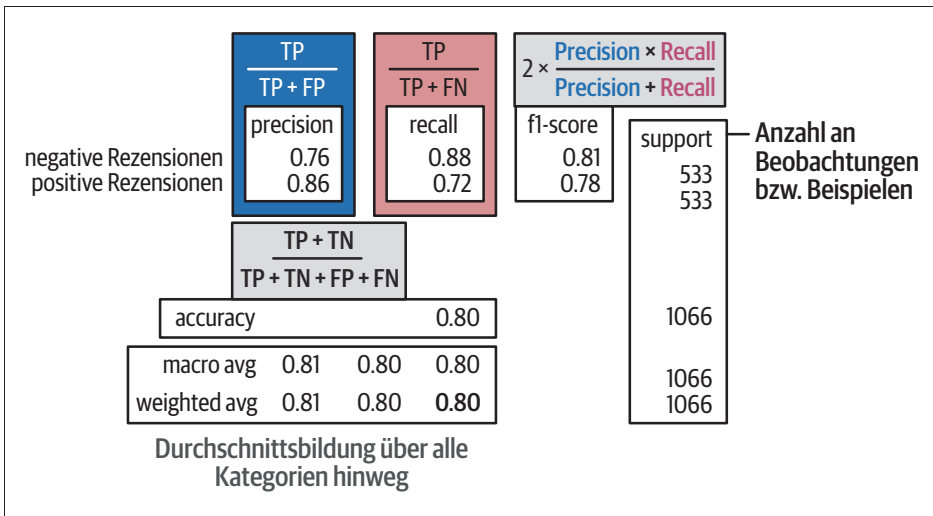


Abbildung 4.9: Der Klassifikationsbericht umfasst mehrere Maße, anhand deren sich die Performance eines Modells bewerten lässt.

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 finetuned 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).

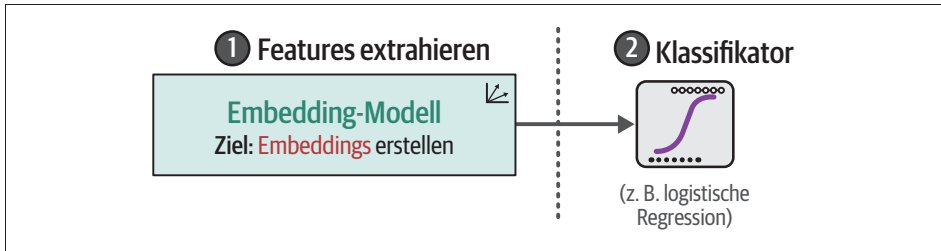


Abbildung 4.10: Die Feature Extraction (erster Schritt) und die Klassifikation (zweiter Schritt) erfolgen separat.

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.

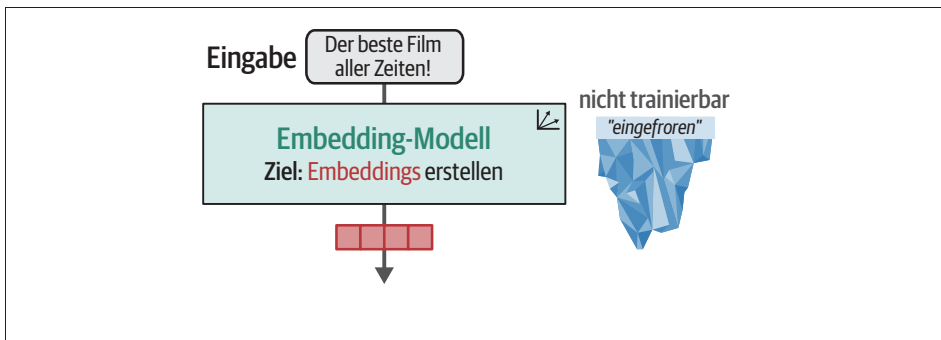


Abbildung 4.11: Im ersten Schritt wird das Embedding-Modell verwendet, um Features zu extrahieren und den Eingabetext in Embeddings umzuwandeln.

Dieser Schritt kann mit dem `sentence-transformers`-Paket durchgeführt werden, das häufig zur Nutzung vortrainierter Embedding-Modelle herangezogen wird.<sup>6</sup> 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.

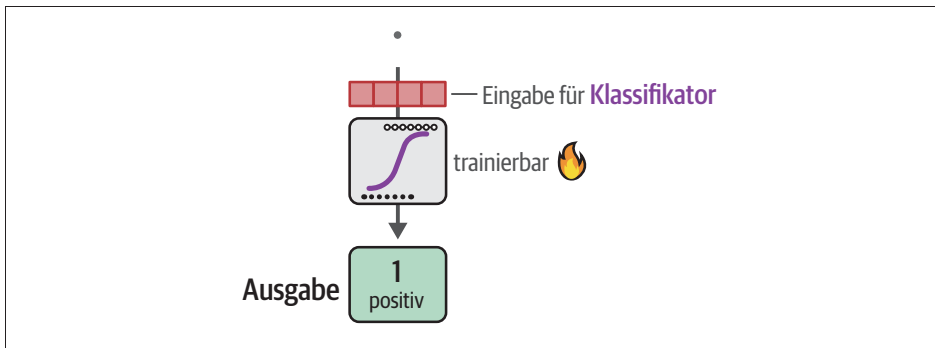


Abbildung 4.12: Auf Basis der als Input-Features verwendeten Embeddings wird ein logistisches Regressionsmodell auf den Trainingsdaten trainiert.

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 unbekannt) 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.

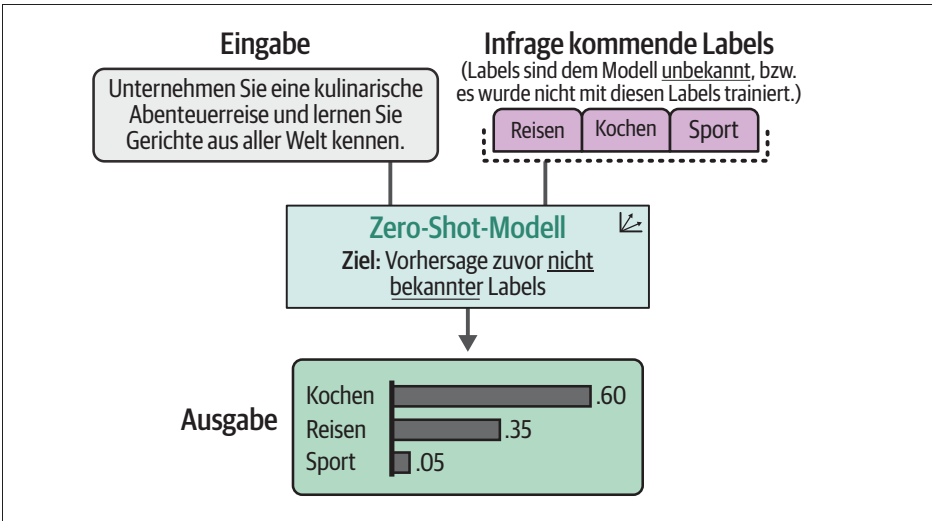


Abbildung 4.13: Bei der Zero-Shot-Klassifikation liegen keine gelabelten Daten vor. Es ist lediglich bekannt, welche Labels bzw. Kategorien in die Klassifikation einbezogen werden sollen. Das Zero-Shot-Modell beurteilt, welchem der infrage kommenden Labels die eingegebenen Daten jeweils zugeordnet werden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

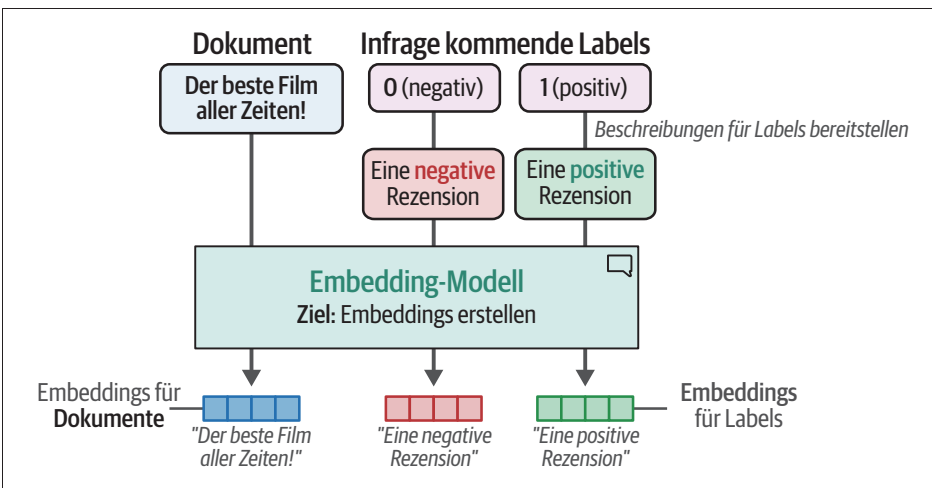


Abbildung 4.14: Um Embeddings für die Labels zu erstellen, ist es zunächst notwendig, deren Bedeutung zu beschreiben, beispielsweise durch Formulierungen wie »Eine negative Rezension« oder »Das ist eine Filmrezension, die eine negative Bewertung zum Ausdruck bringt« (bzw. auf Englisch, wenn die Beispiele 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.

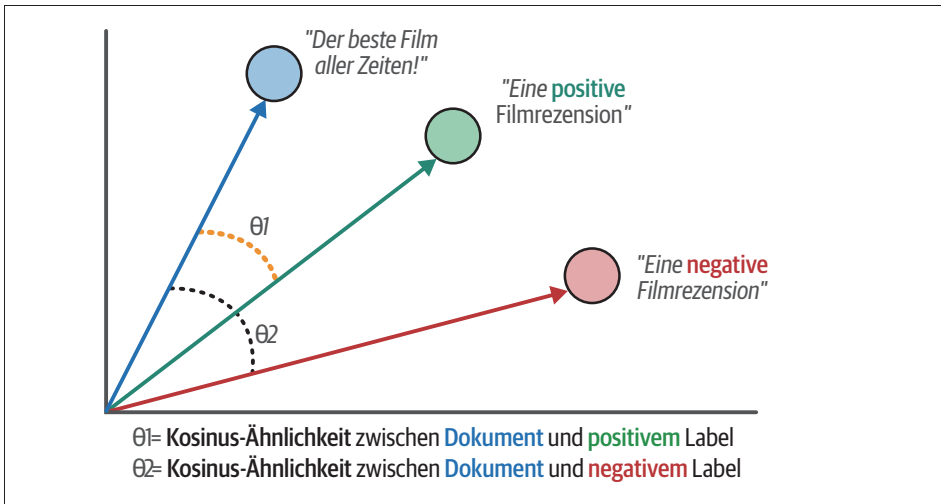


Abbildung 4.15: Die Kosinus-Ähnlichkeit entspricht dem Winkel zwischen zwei Vektoren bzw. Embeddings. Im vorliegenden Beispiel wird jeweils die Ähnlichkeit zwischen einem Dokument und den beiden infrage kommenden Labels (positiv und negativ) ermittelt.

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).

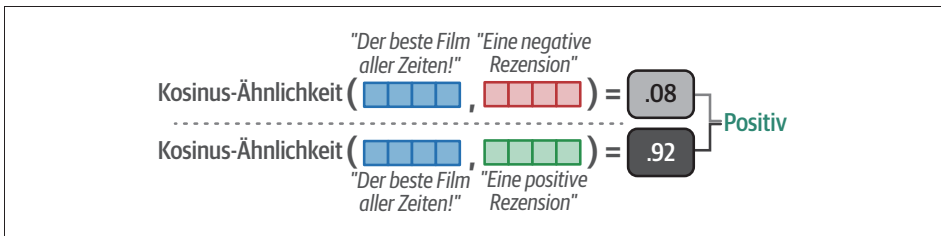


Abbildung 4.16: Nachdem Sie die Embeddings für die Beschreibungen der Labels und die Dokumente erstellt haben, können Sie die Kosinus-Ähnlichkeit zwischen den Labels und den Dokumenten berechnen.

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).

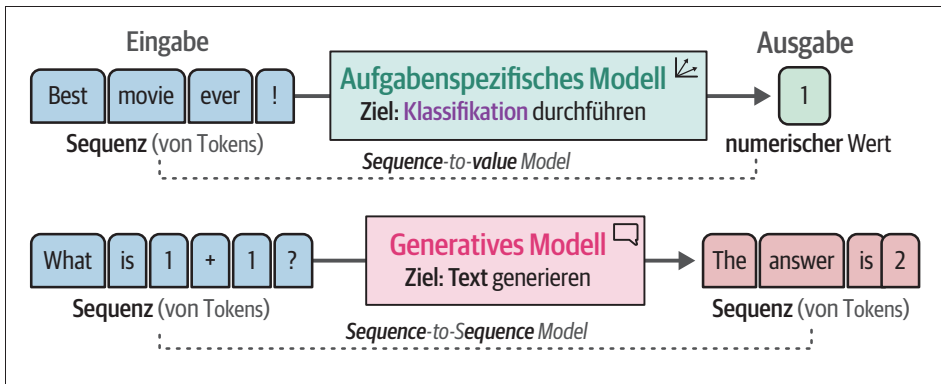


Abbildung 4.17: Ein aufgabenspezifisches Modell gibt infolge einer vorgegebenen Sequenz von Tokens einen numerischen Wert aus (»Sequence-to-Value«-Modell). Ein generatives Modell generiert hingegen infolge einer vorgegebenen Sequenz von Tokens eine weitere Sequenz von Tokens (»Sequence-to-Sequence«-Modell).

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.

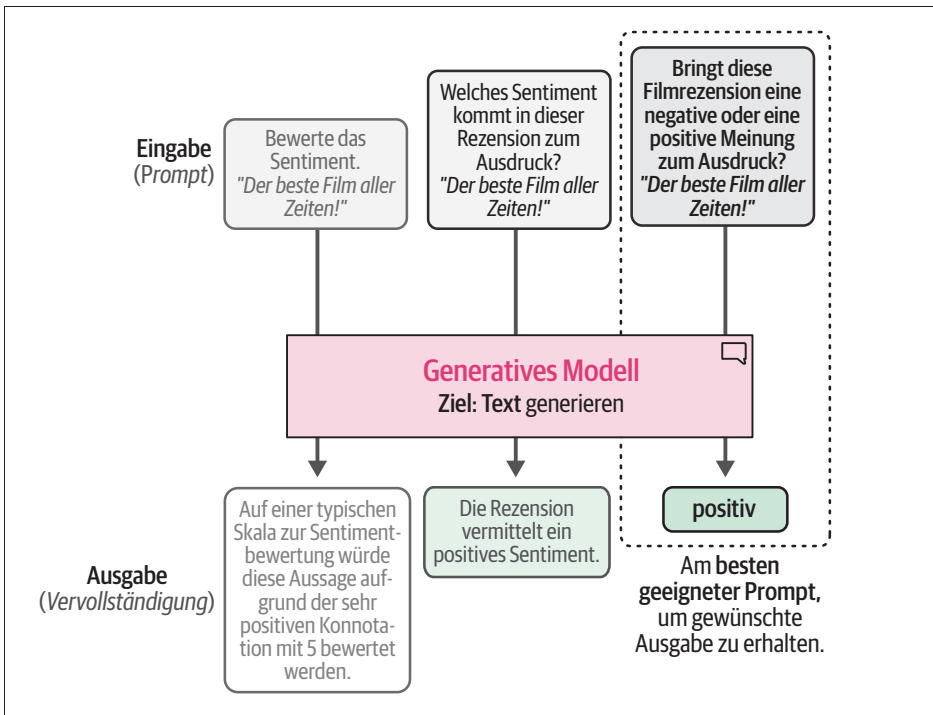


Abbildung 4.18: Im Rahmen des sogenannten Prompt Engineering wird der Prompt nach und nach angepasst, um die vom Modell generierte Ausgabe zu optimieren.

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).<sup>7</sup>

<sup>7</sup> 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.

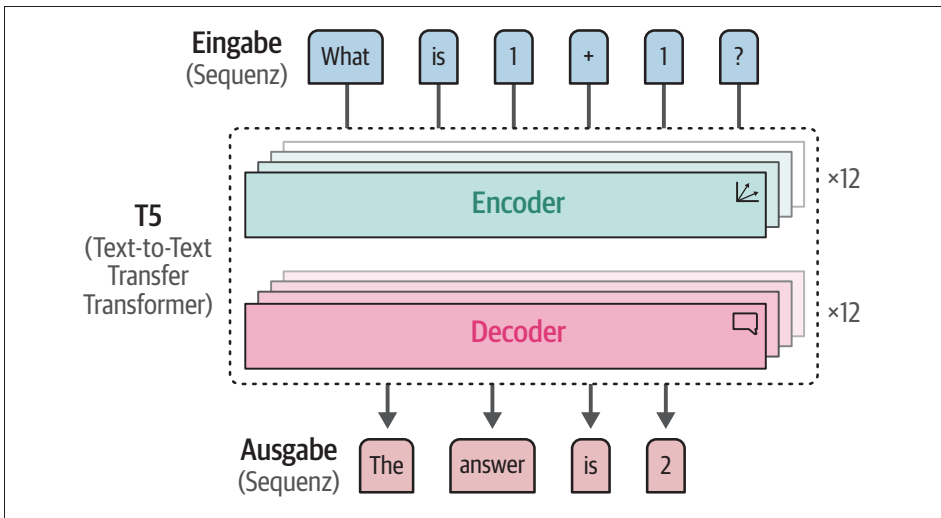


Abbildung 4.19: Die Architektur von T5-Modellen entspricht im Prinzip der des ursprünglichen Transformer-Modells, also einer Decoder-Encoder-Architektur.

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.

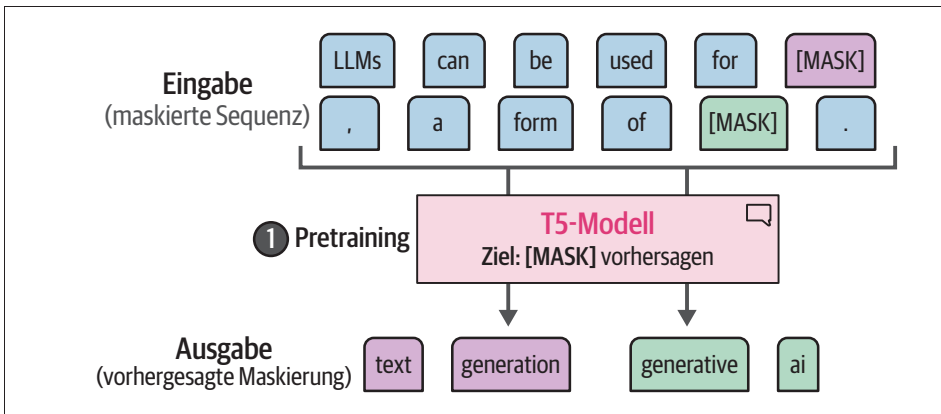


Abbildung 4.20: Im ersten Trainingsschritt, dem Pretraining, muss das T5-Modell maskierte Stellen vorhersagen, an denen sich potenziell mehrere Tokens befinden können.

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).

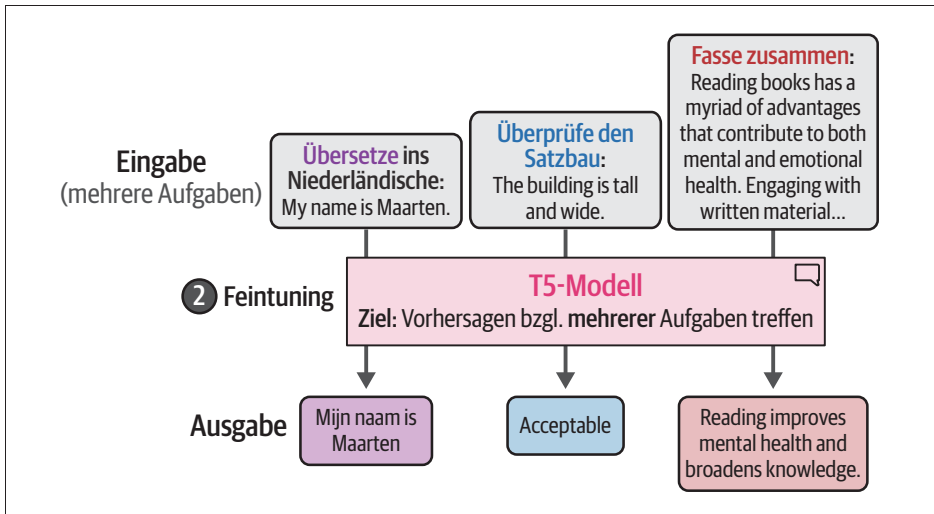


Abbildung 4.21: Da die einzelnen Aufgaben in Anweisungen (in Textform) übertragen werden, kann das T5-Modell während des Feintunings für eine Vielzahl von Aufgaben trainiert werden.

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.<sup>8</sup> 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/x1/xx1). 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.

<sup>8</sup> 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 dienen dann als Grundlage für die Erstellung einer ersten Modellvariante (siehe Abbildung 4.22).

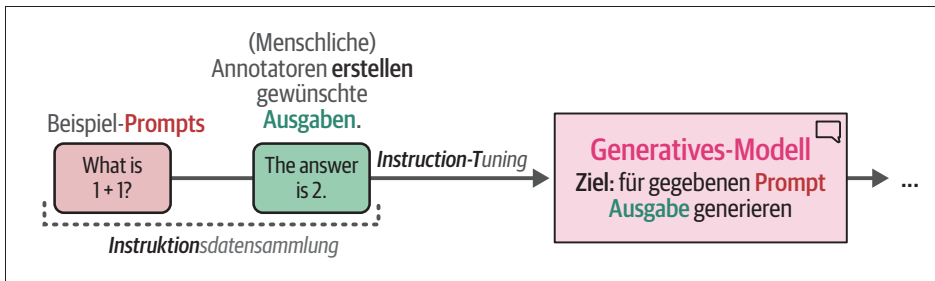


Abbildung 4.22: Für das Feintuning (Instruction Tuning) wurden manuell gelabelte Daten verwendet. Diese bestehen jeweils aus einer Anweisung (Prompt) und der entsprechenden gewünschten Ausgabe.

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.

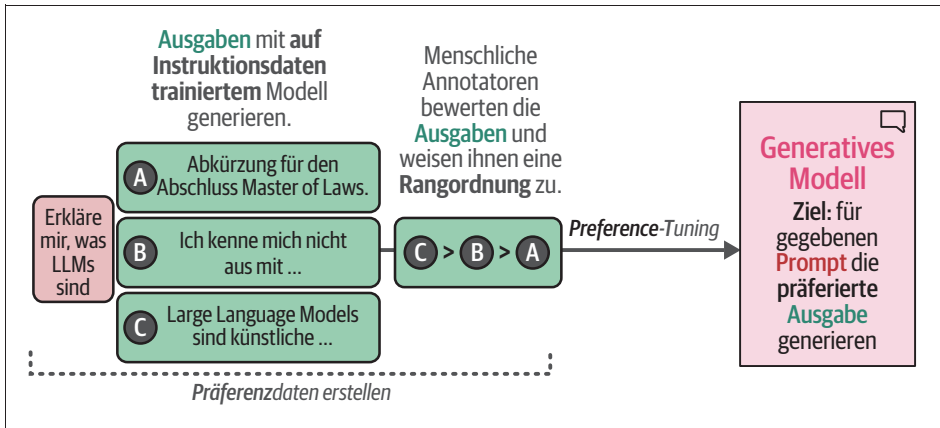


Abbildung 4.23: Zur Erstellung des endgültigen ChatGPT-Modells wurden Präferenzdaten verwendet, also von Menschen bewertete Daten, aus denen hervorging, welche Ausgaben jeweils bevorzugt wurden.

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, welche leistungsfähigen 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-

deutliche, 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 dienen.

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.