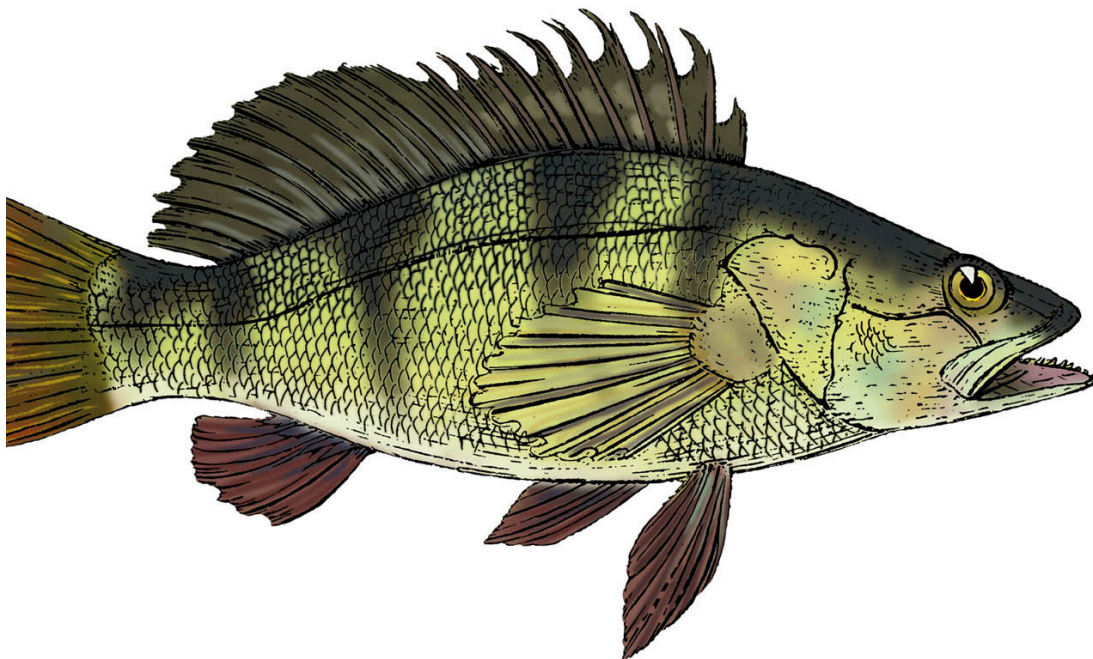


O'REILLY®

Deutsche
Ausgabe

Praxiseinstieg Large Language Models

Strategien und Best Practices für den
Einsatz von ChatGPT und anderen LLMs



Sinan Ozdemir

Übersetzung von Frank Langenau

Inhalt

Cover

Lob für »Praxiseinstieg Large Language Models«

Titel

Impressum

Inhalt

Vorwort

Einleitung

Teil I: Einführung in Large Language Models

1 Überblick über Large Language Models

Was sind Large Language Models?

Definition von LLMs

Hauptmerkmale von LLMs

Wie LLMs funktionieren

Gängige moderne LLMs

BERT

GPT-3 und ChatGPT

T5

Domänenspezifische LLMs

Anwendungen von LLMs

Klassische NLP-Aufgaben

Freitexterzeugung

Informationsabruf/neuronale semantische Suche

Chatbots

Zusammenfassung

2 Semantische Suche mit LLMs

Die Aufgabe

Asymmetrische semantische Suche

Die Lösung im Überblick

Die Komponenten

Engines für Text-Embeddings

Chunking von Dokumenten

Vektordatenbanken

Pinecone

Open-Source-Alternativen

Neueinstufen der abgerufenen Ergebnisse

API

Alles zusammen

Performance

Die Kosten von Closed-Source-Komponenten

Zusammenfassung

3 Erstes Prompt Engineering und ein Chatbot mit ChatGPT

Prompt Engineering

Ausrichtung in Sprachmodellen

Einfach fragen

Few-Shot-Learning

Strukturierung der Ausgabe

Personas fordern auf

Mit Prompts modellübergreifend arbeiten

ChatGPT

Cohere

Open-Source-Prompt-Engineering

Einen Frage-Antwort-Bot mit ChatGPT aufbauen

Zusammenfassung

Teil II: Das Beste aus LLMs herausholen

4 LLMs mit individuellem Feintuning optimieren

Transfer Learning und Feintuning: die Grundlagen

Der Feintuning-Prozess im Detail

Vortrainierte Closed-Source-Modelle als Grundlage

Die OpenAI-API für das Feintuning

Die GPT-3-API für das Feintuning

Fallstudie 1: Stimmungsklassifizierung von Amazon-Rezensionen

Richtlinien und bewährte Methoden für Daten

Individuelle Beispiele mit der OpenAI-CLI vorbereiten

Die OpenAI-CLI einrichten

Hyperparameter auswählen und optimieren

Unser erstes feingetuntes LLM

Feingetunte Modelle mit quantitativen Metriken bewerten

Qualitative Bewertungstechniken

Feingetunte GPT-3-Modelle in Anwendungen integrieren

Fallstudie 2: Klassifizierung der Kategorien von Amazon-Rezensionen

Zusammenfassung

5 Fortgeschrittenes Prompt Engineering

Prompt-Injection-Angriffe

Eingaben und Ausgaben validieren

Beispiel: Validierungspipelines mit NLI aufbauen

Prompts im Stapel verarbeiten

Prompts verketteten

Verkettung als Schutz gegen Prompt Injection

Verkettung, um Prompt Stuffing zu verhindern

Beispiel: Sicherheit durch Verkettung multimodaler LLMs

Prompting mit Gedankenkette

Beispiel: Grundlegende Arithmetik

 Noch einmal: Few-Shot-Learning

Beispiel: Grundschularithmetik mit LLMs

 Testen und iterative Entwicklung von Prompts

 Zusammenfassung

 6 Embeddings und Modellarchitekturen anpassen

 Fallstudie: Ein Empfehlungssystem aufbauen

Das Problem und die Daten einrichten

Das Problem der Empfehlung definieren

Unser Empfehlungssystem im Überblick

Ein benutzerdefiniertes Beschreibungsfeld generieren, um Artikel zu vergleichen

Mit Basis-Embeddern eine Baseline einrichten

Die Feintuning-Daten vorbereiten

Open-Source-Embedder mithilfe von Sentence Transformers feintunen

Zusammenfassung der Ergebnisse

 Zusammenfassung

Teil III: Fortgeschrittene LLM-Nutzung

 7 Jenseits der Basismodelle: LLMs kombinieren

 Fallstudie: Visuelles Frage-Antwort-System

Einführung in unsere Modelle: der Vision Transformer, GPT-2 und DistilBERT

Projektion und Fusion verborgener Zustände

Was ist Cross-Attention, und warum ist sie entscheidend?

Unser benutzerdefiniertes multimodales Modell

Unsere Daten: Visual QA

Die VQA-Trainingsschleife

Zusammenfassung der Ergebnisse

 Fallstudie: Reinforcement Learning from Feedback

Unser Modell: FLAN-T5

Unser Belohnungsmodell: Sentiment und grammatische Korrektheit

Die Bibliothek Transformer Reinforcement Learning

Die RLF-Trainingsschleife

Zusammenfassung der Ergebnisse

Zusammenfassung

8 Feintuning fortgeschrittener Open-Source-LLMs

Beispiel: Multilabel-Klassifizierung mit BERT für Anime-Genres

Die Performance für die Multilabel-Genre-Vorhersage von Anime-Titeln mit dem Jaccard-Koeffizienten messen

Eine einfache Feintuning-Schleife

Allgemeine Tipps zum Feintuning von Open-Source-LLMs

Zusammenfassung der Ergebnisse

Beispiel: LaTeX-Generierung mit GPT-2

Prompt Engineering für Open-Source-Modelle

Zusammenfassung der Ergebnisse

SAWYER: Sinans Versuch, kluge und dennoch fesselnde Antworten zu geben

Schritt 1: Überwachtes Feintuning mit Anweisungen

Schritt 2: Training des Belohnungsmodells

Schritt 3: Reinforcement Learning mit (geschätzter) menschlicher Rückkopplung

Zusammenfassung der Ergebnisse

Die sich ständig verändernde Welt des Feintunings

Zusammenfassung

9 LLMs in die Produktion überführen

Closed-Source-LLMs in der Produktion bereitstellen

Kostenprognosen

API-Schlüsselverwaltung

Open-Source-LLMs in der Produktion bereitstellen

Ein Modell für Inferenz vorbereiten

Interoperabilität

Quantisierung

Beschneiden

Wissensdestillation

Fallstudie: Unsere Anime-Genre-Vorhersage destillieren

Kostenprognosen mit LLMs

Die Plattform Hugging Face

Zusammenfassung

Ihre Beiträge sind wichtig

Weitermachen!

Teil IV: Anhänge

Anhang A: LLM-FAQs

Anhang B: LLM-Glossar

Anhang C: Archetypen von LLM-Anwendungen

Index

Über den Autor

Kolophon

LLMs mit individuellem Feintuning optimieren

Bisher haben wir ausschließlich LLMs – sowohl Open Source als auch Closed Source – verwendet, da sie in einsatzbereiten Versionen verfügbar sind. Wir haben uns auf die Leistungsfähigkeit der Attention-Mechanismen von Transformern und deren Rechengeschwindigkeit verlassen, um einige ziemlich komplexe Probleme relativ leicht lösen zu können. Wie Sie sich wahrscheinlich schon denken, ist das nicht immer genug.

In diesem Kapitel tauchen wir ein in die Welt des Feintunings großer Sprachmodelle (LLMs), um deren volles Potenzial zu erschließen. Durch Feintuning werden Standardmodelle aktualisiert und in die Lage versetzt, hochwertigere Ergebnisse zu erzielen. Dabei kann es zur Einsparung von Token und oftmals geringeren Anforderungen an die Latenz kommen. Während das Vortraining von GPT-ähnlichen LLMs mit umfangreichen Textdaten beeindruckende Fähigkeiten beim Few-Shot-Learning ermöglicht, geht Feintuning einen Schritt weiter, indem das Modell anhand einer Vielzahl von Beispielen feingetunt wird, was über verschiedenartige Aufgaben hinweg zu einer überlegenen Leistung führt.

Inferenz mit feingetunten Modellen durchzuführen, kann auf lange Sicht äußerst kosteneffektiv sein, insbesondere wenn man mit kleineren Modellen arbeitet. Zum Beispiel kostet ein feingetuntes ADA-Modell von OpenAI (mit lediglich 350 Millionen Parametern) nur 0,0016 Dollar pro 1.000 Token, während ChatGPT (1,5 Milliarden Parameter) mit 0,002 Dollar und DaVinci (175 Milliarden Parameter) mit 0,002 Dollar zu Buche schlagen. Im Laufe der Zeit werden die Kosten für die Verwendung eines feingetunten Modells wesentlich attraktiver, wie Abbildung 4-1 veranschaulicht.

In diesem Kapitel führe ich Sie durch den Prozess des Feintunings, angefangen mit der Vorbereitung der Trainingsdaten über Strategien für das Training eines neuen oder bestehenden feingetunten Modells bis hin zu einer Diskussion darüber, wie Sie Ihr feingetuntes Modell in reale Anwendungen einbinden können. Da dies ein umfangreiches Thema ist, müssen wir davon ausgehen, dass einige wichtige Aufgaben hinter den Kulissen erledigt werden, zum Beispiel das Beschriften der Daten. In vielen Fällen komplexer und spezifischer Aufgaben kann das Beschriften der Daten einen hohen Aufwand bedeuten. Fürs Erste nehmen wir deshalb an, dass wir uns größtenteils auf die Beschriftungen in unseren Daten verlassen können. Weitere Informationen darüber, wie Sie in derartigen Fällen vorgehen, finden Sie in meinen anderen Beiträgen über Feature Engineering und Label Cleaning.

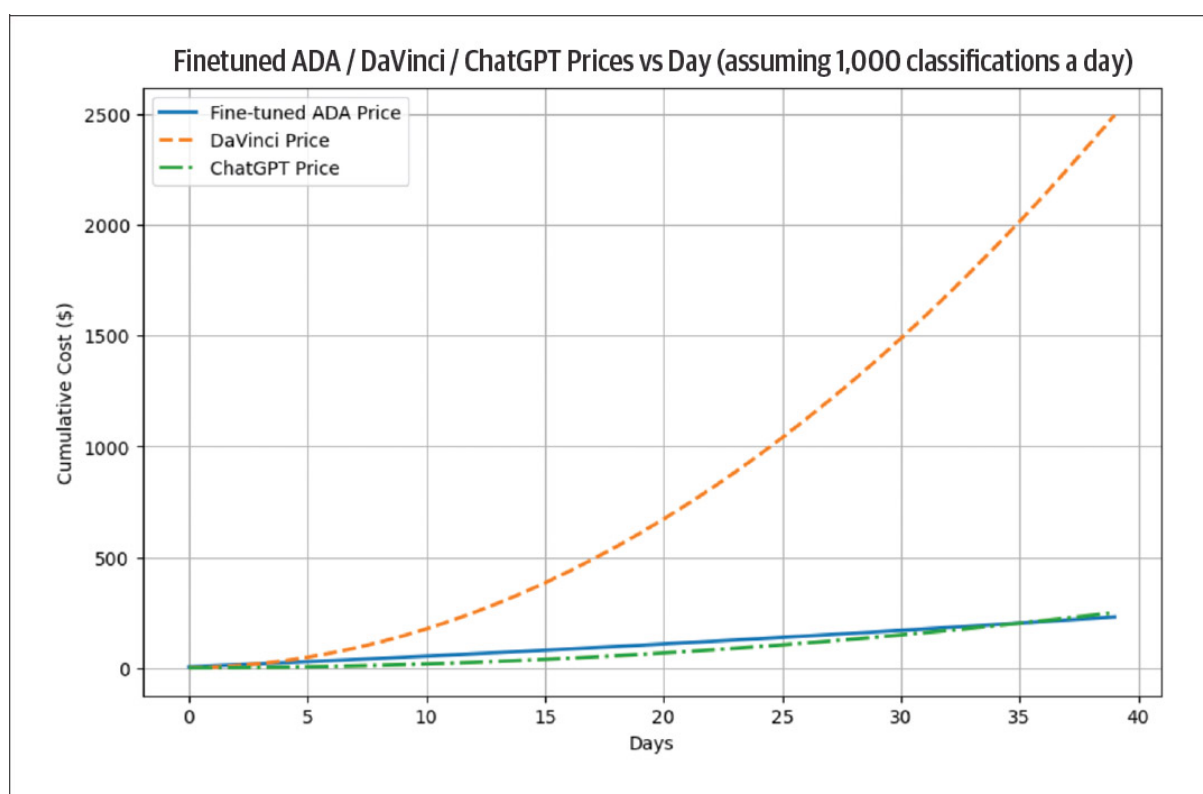


Abbildung 4-1: Geht man von nur 1.000 Klassifizierungen pro Tag und einem relativ großzügigen Prompt-Verhältnis aus (150 Token – für Few-Shot-Beispiele, Anweisungen und andere Elemente – für DaVinci oder ChatGPT pro 40 Token), sind die Kosten für ein feingetuntes Modell, selbst mit einem Anteil an Vorabkosten, fast immer günstiger als die Gesamtkosten pro Tag. Allerdings sind hier nicht die Kosten für das Feintuning eines Modells berücksichtigt, die wir später in diesem Kapitel untersuchen werden.

Wenn Sie die Nuancen des Feintunings verstehen und die entsprechenden Techniken beherrschen, sind Sie gut gerüstet, um sich die Leistungsfähigkeit von

LLMs nutzbar zu machen und maßgeschneiderte Lösungen für Ihre konkreten Bedürfnisse zu entwickeln.

Transfer Learning und Feintuning: die Grundlagen

Feintuning ist von der Idee des Transfer Learning abhängig. *Transfer Learning* (Transferlernen) ist eine Technik, die vortrainierte Modelle nutzt, um auf vorhandenem Wissen für neue Aufgaben oder Bereiche aufzubauen. Im Fall von LLMs beinhaltet dies die Nutzung des Vortrainings, um das allgemeine Sprachverständnis, einschließlich Grammatik und Allgemeinwissen, auf bestimmte domänenspezifische Aufgaben zu übertragen. Allerdings kann das Vortraining nicht genügen, um die Nuancen bestimmter geschlossener oder spezialisierter Themen zu verstehen, etwa bei der rechtlichen Struktur oder den Richtlinien eines Unternehmens.

Feintuning ist eine spezielle Form des Transfer Learning, bei der die Parameter eines vortrainierten Modells angepasst werden, um einer »nachgelagerten« Zielaufgabe besser gerecht zu werden. Durch das Feintuning können LLMs aus individuellen Beispielen lernen und relevante und genaue Antworten effizienter generieren.

Der Feintuning-Prozess im Detail

Zum Feintuning eines Deep-Learning-Modells aktualisiert man die Parameter des Modells, um seine Performance bei einer bestimmten Aufgabe oder einem Datenset zu verbessern.

- **Trainingsdatenset:** Eine Sammlung von beschrifteten Beispielen, mit denen das Modell trainiert wird. Das Modell lernt, Muster und Beziehungen in den Daten zu erkennen, indem seine Parameter anhand der Trainingsbeispiele angepasst werden.
- **Validierungsdatenset:** Eine separate Sammlung von beschrifteten Beispielen, anhand deren die Performance des Modells während des Trainings bewertet wird.
- **Testdatenset:** Eine dritte Sammlung von beschrifteten Beispielen, die sowohl vom Trainings- als auch vom Validierungsdatenset getrennt ist. Dieses Datenset dient dazu, die endgültige Performance des Modells zu bewerten, nachdem Training und Feintuning abgeschlossen sind. Das Testdatenset liefert eine endgültige, unvoreingenommene Schätzung der Fähigkeit des Modells, neue, bisher ungesehene Daten verallgemeinern zu können.

- **Verlustfunktion:** Eine Funktion, die die Differenz zwischen den Modellvorhersagen und den tatsächlichen Zielwerten quantifiziert. Das Ergebnis dient als Fehlermaß, um die Performance des Modells zu bewerten und den Optimierungsprozess zu steuern. Ziel des Trainings ist es, die Verlustfunktion zu minimieren, um bessere Vorhersagen zu erzielen.

Der Prozess des Feintunings lässt sich in folgende Schritte aufgliedern:

1. **Beschriftete Daten sammeln:** Der erste Schritt beim Feintuning besteht darin, beschriftete Beispiele, die für die Zielstellung oder den Bereich relevant sind, für Trainings-, Validierungs- und Testdatensets zu sammeln. Beschriftete Daten dienen dem Modell als Orientierung, um die aufgabenspezifischen Muster und Beziehungen zu lernen. Wenn das Modell zum Beispiel für die Klassifizierung von Gefühlen (unser erstes Beispiel) feingetunt werden soll, müsste das Datenset Textbeispiele zusammen mit den jeweiligen Gefühlsbezeichnungen wie positiv, negativ oder neutral enthalten.
2. **Hyperparameter auswählen:** Beim Feintuning werden Hyperparameter angepasst, die den Lernprozess beeinflussen – zum Beispiel Lernrate, Stapelgröße und Anzahl der Epochen. Die Lernrate bestimmt die Schrittweite bei der Aktualisierung der Modellgewichte, während sich die Stapelgröße auf die Anzahl der Trainingsbeispiele bezieht, die in einer einzelnen Aktualisierung verwendet werden. Die Anzahl der Epochen gibt an, wie viele Male das Modell über das gesamte Trainingsdatenset iteriert. Wenn man diese Hyperparameter richtig einstellt, kann sich dies erheblich auf die Performance des Modells auswirken und dazu beitragen, Probleme wie Überanpassung und Unteranpassung zu vermeiden. Bei Überanpassung lernt ein Modell mehr das Rauschen in den Trainingsdaten als die eigentlichen Signale, während das Modell bei Unteranpassung nicht in der Lage ist, die zugrunde liegende Struktur der Daten zu erfassen.
3. **Modell anpassen:** Nachdem die beschrifteten Daten und Hyperparameter festgelegt sind, muss das Modell gegebenenfalls an die Zielaufgabe angepasst werden. Hierzu ist es erforderlich, die Modellarchitektur zu modifizieren, beispielsweise benutzerdefinierte Ebenen hinzuzufügen oder die Ausgabestruktur zu ändern, um der Zielaufgabe besser zu entsprechen. Zum Beispiel kann die Architektur von BERT von Haus aus keine Sequenzklassifizierung durchführen, doch man kann die Architektur recht leicht modifizieren, um diese Aufgabe zu bewerkstelligen. In unserer Fallstudie können wir dieses Thema

aussparen, weil OpenAI das für uns übernimmt. In einem späteren Kapitel werden wir uns aber damit befassen müssen.

4. **Bewerten und iterieren:** Nachdem das Feintuning abgeschlossen ist, müssen wir die Performance des Modells mit einem separaten, zurückgehaltenen (Holdout-)Validierungsdatenset bewerten, um sicherzustellen, dass es ungesehene Daten gut verallgemeinern kann. Hierfür lassen sich je nach Aufgabe Performancemetriken wie Genauigkeit, F1-Maß oder mittlerer absoluter Fehler (*Mean Absolute Error*, MAE) verwenden. Wenn die Performance nicht zufriedenstellend ist, sind möglicherweise Anpassungen der Hyperparameter oder des Datensets erforderlich, was ein erneutes Training des Modells nach sich zieht.
5. **Modell implementieren und weitertrainieren:** Sobald das Modell feingetunt ist und wir mit seiner Performance zufrieden sind, müssen wir es in bestehende Infrastrukturen integrieren, sodass wir mit eventuellen Fehlern umgehen und Feedback von Benutzern einholen können. Auf diese Weise können wir unser gesamtes Datenset erweitern und den Prozess in Zukunft wiederholen.

Abbildung 4-2 skizziert diesen Prozess. Der beschriebene Prozess kann mehrere Iterationen und eine sorgfältige Abwägung von Hyperparametern, Datenqualität und Modellarchitektur umfassen, um die gewünschten Ergebnisse zu erzielen.

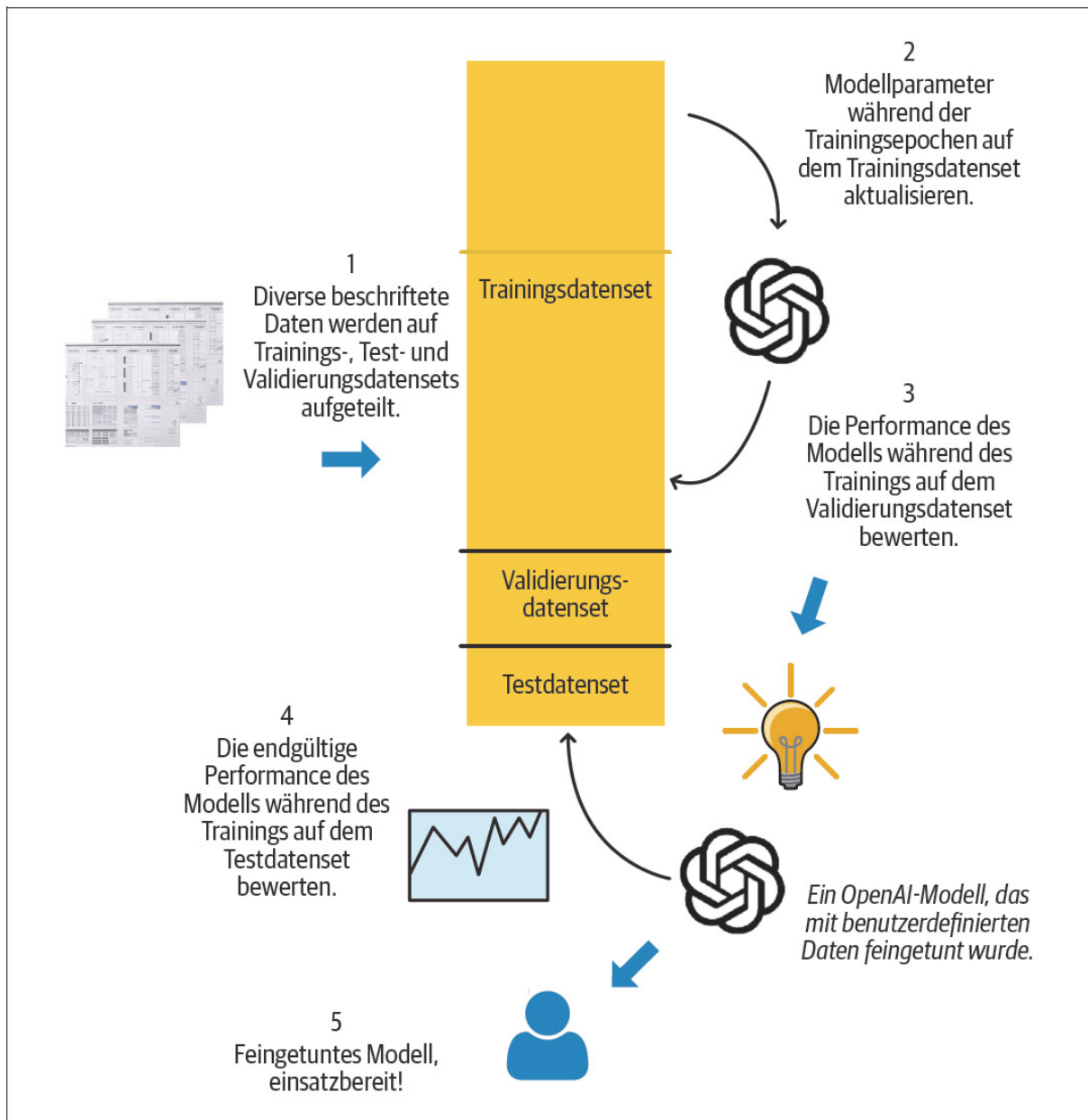


Abbildung 4-2: Übersicht über den Feintuning-Prozess. Ein Datenset wird in Trainings-, Validierungs- und Testdatensets aufgeteilt. Das Trainingsdatenset dient dazu, die Gewichte des Modells zu aktualisieren und das Modell zu bewerten, während das Validierungsdatenset verwendet wird, um das Modell während des Trainings zu bewerten. Das endgültige Modell wird dann mit dem Testdatenset getestet und anhand einer Reihe von Kriterien bewertet. Besteht das Modell alle diese Tests, wird es in der Produktion eingesetzt und für weitere Iterationen überwacht.

Vortrainierte Closed-Source-Modelle als Grundlage

Vortrainierte LLMs spielen eine entscheidende Rolle beim Transfer Learning und Feintuning, da sie eine Grundlage für das allgemeine Sprachverständnis und -

wissen bieten. Diese Grundlage ermöglicht eine effiziente Anpassung der Modelle an spezifische Aufgaben und Wissensbereiche und verringert den Bedarf an umfangreichen Trainingsressourcen und -daten.

Im Mittelpunkt dieses Kapitels steht das Feintuning von LLMs mithilfe der Infrastruktur von OpenAI, die speziell dafür konzipiert wurden, diesen Prozess zu erleichtern. OpenAI hat Tools und Ressourcen entwickelt, um es Forscherinnen und Entwicklern zu erleichtern, kleinere Modelle wie Ada und Babbage auf ihre spezifischen Bedürfnisse abzustimmen. Die Infrastruktur bietet einen rationellen Ansatz für das Feintuning, der es Benutzern ermöglicht, bereits trainierte Modelle effizient an ein breites Spektrum von Aufgaben und Fachbereichen anzupassen.

Vorteile der OpenAI-Infrastruktur für das Feintuning

Die OpenAI-Infrastruktur für das Feintuning bietet mehrere Vorteile:

- Zugriff auf leistungsfähige vortrainierte Modelle wie GPT-3, die auf umfangreichen und vielfältigen Datensätzen trainiert wurden.
- Eine relativ benutzerfreundliche Schnittstelle, die das Feintuning für Personen mit unterschiedlichem Wissenstand vereinfacht.
- Eine Reihe von Tools und Ressourcen, die den Nutzerinnen und Nutzern helfen, ihren Feintuning-Prozess zu optimieren, wie zum Beispiel Richtlinien für die Auswahl von Hyperparametern, Tipps für die Vorbereitung von individuellen Beispielen und Ratschläge für die Modellbewertung.

Dieser rationelle Prozess spart Zeit und Ressourcen und gewährleistet gleichzeitig die Entwicklung hochwertiger Modelle, die genaue und relevante Antworten für ein breites Spektrum von Anwendungen generieren können. In den Kapiteln 6 und 9 befassen wir uns eingehender mit dem Feintuning von Open-Source-Modellen und deren Vor- und Nachteilen.

Die OpenAI-API für das Feintuning

Die GPT-3-API dient dem Zugriff auf eines der fortschrittlichsten LLMs auf dem Markt. Diese API bietet eine Reihe von Funktionen für das Feintuning, sodass Entwickler das Modell an spezifische Aufgaben, Sprachen und Fachbereiche anpassen können. In diesem Abschnitt geht es um die wichtigsten Features der GPT-3-API für das Feintuning, die unterstützten Methoden und bewährte Verfahren für ein erfolgreiches Feintuning von Modellen.

Die GPT-3-API für das Feintuning

Die GPT-3-API für das Feintuning ist wie eine Schatztruhe voller leistungsstarker Features, die das Anpassen des Modells zu einem Kinderspiel machen. Von der Unterstützung verschiedener Fähigkeiten für das Feintuning bis zu einer Palette von Methoden ist die API eine zentrale Anlaufstelle, wenn es um die Anpassung des Modells an Ihre spezifischen Aufgaben, Sprachen oder Bereiche geht. Dieser Abschnitt lüftet die Geheimnisse der GPT-3-API für das Feintuning und stellt Tools und Techniken vor, die sie zu einer unschätzbar wertvollen Ressource machen.

Fallstudie 1: Stimmungsklassifizierung von Amazon-Rezensionen

In unserer ersten Fallstudie arbeiten wir mit dem Datenset `amazon_reviews_multi` (siehe Abbildung 4-3). Dieses Datenset ist eine Sammlung von Produktrezensionen von Amazon, die mehrere Produktkategorien und Sprachen (Englisch, Japanisch, Deutsch, Französisch, Chinesisch und Spanisch) abdecken. Zu jeder Rezension im Datenset gehört eine Bewertung auf einer Skala von einem bis fünf Sternen, wobei ein Stern die niedrigste und fünf Sterne die höchste Bewertung darstellen. Mit dieser Fallstudie verfolgen wir das Ziel, ein vortrainiertes Modell von OpenAI zu optimieren, damit sich damit eine Stimmungsklassifizierung dieser Rezensionen durchführen und die Anzahl der in einer Rezension vergebenen Sterne vorhersagen lässt. Nehmen wir eine Seite aus meinem eigenen Buch und beginnen wir mit der Analyse der Daten.

review_title	review_body	stars
Did not work on Galaxy S9	I plugged the cord into my Galaxy S9 and I kep...	1
Zufrieden	Der Stuhl ist super gemütlich und war auch seh...	4
译的问题, 不能理解到精髓	还可以的一本书, 可能是翻译的问题, 不能理解到精髓	4
bien mais pas plus	pour ma petite fille qui adore Minie	3

Sechs Sprachen in 1,2 Millionen Zeilen

Titel und Text zusammen bilden den vollständigen Rezensionskontext.

Unsere vorherzusagende Klasse (die Antwort).

Abbildung 4-3: Ein Ausschnitt aus dem Datenset »amazon_reviews_multi« zeigt den Eingabekontext (Titel und Text der Rezensionen) und die Antwort (die Sache, die wir vorhersagen wollen – die Anzahl der von den Rezensenten vergebenen Sterne).

In dieser Runde des Feintunings kümmern wir uns um die folgenden drei Spalten im Datenset:

- `review_title`: der Titel der Rezension
- `review_body`: der Text der Rezension
- `stars`: eine Ganzzahl zwischen 1 und 5, die die Anzahl der Sterne angibt

Unser Ziel ist es, aus dem Kontext von Rezensionstitel und -text die abgegebene Bewertung vorherzusagen.

Richtlinien und bewährte Methoden für Daten

Bei der Auswahl der Daten für ein Feintuning sind im Allgemeinen einige Punkte zu beachten:

- **Datenqualität:** Sicherstellen, dass die für das Feintuning herangezogenen Daten von hoher Qualität sind, kein Rauschen aufweisen und genau den Zielbereich oder die Aufgabe repräsentieren. So kann das Modell effektiv aus den Trainingsbeispielen lernen.
- **Datenvielfalt:** Sicherstellen, dass das Datenset vielfältig ist und ein breites Spektrum an Szenarios abdeckt, damit das Modell in verschiedenen Situationen gut verallgemeinern kann.
- **Datenausgleich:** Eine ausgewogene Verteilung der Beispiele auf verschiedene Aufgaben und Bereiche hilft, eine Überanpassung und

Verzerrungen in der Performance des Modells zu vermeiden. Dies lässt sich bei unausgewogenen Datensets erreichen, indem man Mehrheitsklassen unterrepräsentiert, Minderheitsklassen überrepräsentiert oder synthetische Daten hinzufügt. Unser Stimmungsbild ist perfekt ausgewogen, weil dieses Datenset aufbereitet wurde – aber sehen Sie sich ein weit schwierigeres Beispiel in unserer Codebasis an, bei dem wir versuchen, die sehr unausgewogene Aufgabe der Kategorienklassifizierung zu klassifizieren.

- **Datenmenge:** Bestimmen der Gesamtmenge der Daten, die für das Feintuning des Modells erforderlich sind. Im Allgemeinen benötigen größere Sprachmodelle wie LLMs umfangreichere Daten, um verschiedenartige Muster effektiv zu erfassen und zu lernen, aber auch kleinere Datensets, wenn das LLM mit genügend ähnlichen Daten vortrainiert wurde. Die genaue Menge der benötigten Daten kann je nach Komplexität der jeweiligen Aufgabe variieren. Jedes Datenset sollte nicht nur umfangreich, sondern auch vielfältig und repräsentativ für den Problemraum sein, um potenzielle Verzerrungen zu vermeiden und eine robuste Performance über einen weiten Bereich von Eingaben zu gewährleisten. Zwar lässt sich mit einer großen Menge an Trainingsdaten die Modellperformance verbessern, doch erhöhen sich dadurch auch die rechen-technischen Ressourcen, die für das Modelltraining und das Feintuning erforderlich sind. Dieser Kompromiss muss im Zusammenhang mit den spezifischen Projektanforderungen und -ressourcen betrachtet werden.

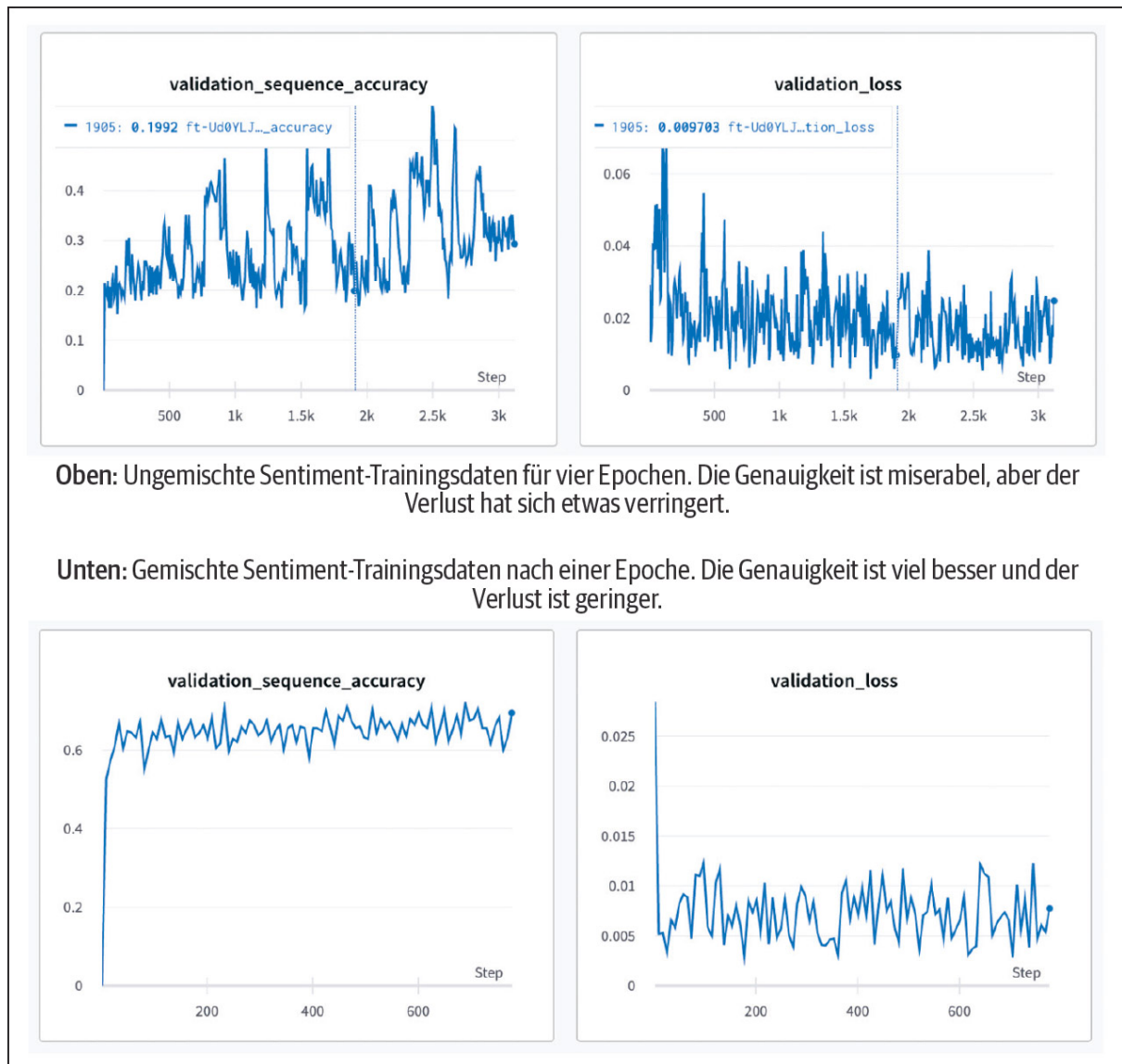
Individuelle Beispiele mit der OpenAI-CLI vorbereiten

Bevor wir uns an das Feintuning machen, müssen wir die Daten vorbereiten, indem wir sie entsprechend den Anforderungen der API bereinigen und formatieren. Dazu gehören die folgenden Schritte:

- **Entfernen von Duplikaten:** Um die höchste Datenqualität sicherzustellen, müssen zunächst alle doppelten Rezensionen aus dem Datenset entfernt werden. Dadurch wird verhindert, dass sich das Modell zu stark an bestimmte Beispiele anpasst, und es verbessert sich seine Fähigkeit, neue Daten zu verallgemeinern.
- **Aufteilung der Daten:** Teilen Sie das Datenset in Trainings-, Validierungs- und Testsets auf, wobei Sie eine zufällige Verteilung der Beispiele über jedes Set beibehalten. Ziehen Sie gegebenenfalls eine geschichtete Zufallsstichprobe (engl. *Stratified Sampling*) in Betracht, um

sicherzustellen, dass jedes Set einen repräsentativen Anteil an verschiedenen Stimmungsbeschriftungen enthält und so die Gesamtverteilung des Datensets erhalten bleibt.

- **Mischen der Trainingsdaten:** Das Mischen der Trainingsdaten vor dem Feintuning hilft, Verzerrungen im Lernprozess zu vermeiden, weil das Modell die Beispiele in einer zufälligen Reihenfolge vorfindet und sich somit das Risiko verringert, unbeabsichtigte Muster basierend auf der Reihenfolge der Beispiele zu lernen. Da das Modell in jeder Trainingsphase einer größeren Vielfalt an Beispielen ausgesetzt wird, kann es später besser verallgemeinern. Außerdem verringert sich das Risiko einer Überanpassung, da sich das Modell weniger wahrscheinlich die Trainingsbeispiele merkt und sich stattdessen auf das Lernen der zugrunde liegenden Muster konzentriert. Abbildung 4-4 skizziert die Vorteile, die das Mischen der Trainingsdaten bietet.



Oben: Ungemischte Sentiment-Trainingsdaten für vier Epochen. Die Genauigkeit ist miserabel, aber der Verlust hat sich etwas verringert.

Unten: Gemischte Sentiment-Trainingsdaten nach einer Epoche. Die Genauigkeit ist viel besser und der Verlust ist geringer.

Abbildung 4-4: Ungemischte Daten sind schlechte Trainingsdaten! Sie geben dem Modell Raum für eine Überanpassung bei bestimmten Datenstapeln und senken die Gesamtqualität der Antworten. Die beiden oberen Diagramme zeigen ein Modell, das mit ungemischten Trainingsdaten trainiert wurde. Die Genauigkeit ist schrecklich im Vergleich zu einem Modell, das mit gemischten Daten trainiert wurde, wie es die beiden unteren Diagramme zeigen.

Im Idealfall werden die Daten vor jeder einzelnen Epoche gemischt, um die Wahrscheinlichkeit einer Überanpassung des Modells an die Daten so weit wie möglich zu verringern.

- **Erzeugen des OpenAI-Formats JSONL:** Die API von OpenAI erwartet, dass die Trainingsdaten im JSONL-Format (newline-delimited JSON) vorliegen. Erstellen Sie dazu in den Trainings- und Validierungsdatensätzen für jedes Beispiel ein JSON-Objekt mit zwei Feldern: `prompt` (die Eingabe) und `completion` (die Zielklasse). Das

Feld `prompt` sollte den Rezensionstext enthalten, und das Feld `completion` nimmt das entsprechende Stimmungslabel (Sterne) auf. Speichern Sie diese JSON-Objekte als Datensätze, die durch eine Zeilenschaltung getrennt sind (newline-delimited), in getrennten Dateien für die Trainings- und Validierungsdatensätze.

Achten Sie bei den Completion-Token in unserem Datenset darauf, dass ein Leerzeichen vor dem Klassenlabel steht. Das Modell erkennt daran, dass es ein neues Token generieren soll. Bei der Vorbereitung der Prompts für den Feintuning-Prozess ist es außerdem nicht notwendig, Few-Shot-Beispiele einzubinden, da das Modell bereits anhand der aufgabenspezifischen Daten feinetunt wurde. Geben Sie stattdessen einen Prompt an, der den Rezensionstext und jeden erforderlichen Kontext enthält, gefolgt von einem Suffix (z.B. »Sentiment:« ohne nachgestellte Leerzeichen oder »\n\n###\n\n« wie in Abbildung 4-5), das das gewünschte Ausgabeformat darstellt. Abbildung 4-5 zeigt ein Beispiel für eine einzelne Zeile unserer JSON-Datei.



Abbildung 4-5: Ein einzelnes JSONL-Beispiel für unsere Trainingsdaten, die wir in OpenAI einspeisen. Jedes JSON-Objekt hat einen Prompt-Schlüssel, der die Eingabe in das Modell ohne Few-Shot-Beispiele, Anweisungen oder andere Daten kennzeichnet, und einen Completion-Schlüssel, der angibt, was das Modell

ausgeben soll – in diesem Fall ein einzelnes Klassifizierungstoken. Im Beispiel bewertet der Benutzer das Produkt mit einem Stern.

Für unsere Eingabedaten habe ich den Titel und den Rezensionstext zu einer einzelnen Eingabe verkettet. Dies war eine persönliche Entscheidung, die meine Überzeugung widerspiegelt, dass der Titel mit einer direkteren Sprache die allgemeine Stimmung anzeigt, während der eigentliche Text wahrscheinlich in einer mehr nuancierten Sprache abgefasst ist, um die genaue Anzahl der Sterne zu bestimmen, die der Rezensent vergeben wird. Probieren Sie ruhig verschiedene Möglichkeiten aus, Textfelder miteinander zu kombinieren! Auf dieses Thema kommen wir in späteren Fallstudien zurück. Dann erläutern wir auch andere Möglichkeiten, Felder für eine einzelne Texteingabe zu formatieren.

Der Code in Beispiel 4-1 lädt das Datenset mit den Amazon-Rezensionen und konvertiert das Teildatenset `train` in einen Pandas-DataFrame. Anschließend wird die benutzerdefinierte Funktion `prepare_df_for_openai` aufgerufen, die den DataFrame vorverarbeitet, d.h. den Titel und den Text der Rezension zu einem Prompt kombiniert, eine neue Completion-Spalte anlegt und den DataFrame filtert, um nur englischsprachige Rezensionen beizubehalten. Schließlich entfernt die Funktion noch die doppelten Zeilen basierend auf der Spalte `prompt` und gibt einen DataFrame zurück, der nur aus den Spalten `prompt` und `completion` besteht.

Beispiel 4-1: Eine JSONL-Datei für die Trainingsdaten unserer Stimmungsanalyse erzeugen

```
from datasets import load_dataset

import pandas as pd

# Das Datenset mit den mehrsprachigen Amazon-Rezensionen laden

dataset = load_dataset("amazon_reviews_multi",
                       "all_languages")

# Das Teildatenset 'train' des Datensets in einen Pandas-
# DataFrame konvertieren

training_df = pd.DataFrame(dataset['train'])
```

```

def prepare_df_for_openai(df):

    # Die Spalten 'review_title' und 'review_body'
    zusammenfassen und

    # ein benutzerdefiniertes Suffix '\n\n###\n\n' am Ende
    hinzufügen,

    # um die Spalte 'prompt' zu erzeugen

    df['prompt'] = df['review_title'] + '\n\n' +
    df['review_body'] + '\n\n###\n\n'

    # Eine neue Spalte 'completion' erzeugen, indem ein
    Leerzeichen vor die

    # 'stars'-Werte gesetzt wird

    df['completion'] = ' ' + df[stars]

    # Den DataFrame filtern, um nur Zeilen mit 'language'
    gleich 'en'

    # (Englisch) zu übernehmen

    english_df = df[df['language'] == 'en']

    # Doppelte Zeilen basierend auf der Spalte 'prompt'
    entfernen

    english_df.drop_duplicates(subset=['prompt'],
    inplace=True)

    # Den gemischten und gefilterten DataFrame nur mit den
    Spalten

```

```

# 'prompt' und 'completion' zurückgeben

return english_df[['prompt',
'completion']].sample(len(english_df))

english_training_df = prepare_df_for_openai(training_df)

# Die Prompts und Completions in eine JSONL-Datei exportieren

english_training_df.to_json("amazon-english-full-train-
sentiment.jsonl", orient='records', lines=True)

```

Ähnlich verfahren wir mit der Teilmenge `validation` des Datensets und der reservierten Teilmenge `test` für einen abschließenden Test des feingetunten Modells. Ein kurzer Hinweis: Zwar filtern wir hier nur nach Englisch, doch es steht Ihnen frei, Ihr Modell mit weiteren Sprachen zu trainieren. Im Beispiel wollte ich einfach ein paar schnelle Ergebnisse zu einem effizienten Preis erhalten.

Die OpenAI-CLI einrichten

Über die OpenAI-Befehlszeilenschnittstelle (*Command Line Interface*, CLI) lässt sich das Feintuning und die Interaktion mit der API einfacher abwickeln. Mit der CLI können Sie Anfragen zum Feintuning absenden, den Trainingsfortschritt überwachen und Ihre Modelle verwalten – und das alles von der Befehlszeile aus. Vergewissern Sie sich, dass Sie die OpenAI-CLI installiert und mit Ihrem API-Schlüssel konfiguriert haben, bevor Sie mit dem Feintuning fortfahren.

Die OpenAI-CLI können Sie mit dem Python-Paketmanager `pip` installieren. Vergewissern Sie sich zunächst, dass Sie Python 3.6 oder höher auf Ihrem System installiert haben. Führen Sie dann die folgenden Schritte aus:

1. Öffnen Sie ein Terminal (unter macOS und Linux) oder eine Eingabeaufforderung (unter Windows).
2. Führen Sie den folgenden Befehl aus, um das Paket `openai` zu installieren:

```
pip install openai
```

Dieser Befehl installiert das OpenAI-Paket von Python, das die CLI enthält.

- Um zu überprüfen, ob die Installation erfolgreich war, führen Sie diesen Befehl aus:

```
openai --version
```

Dieser Befehl sollte die Versionsnummer der installierten OpenAI-CLI anzeigen.

Bevor Sie die OpenAI-CLI verwenden können, müssen Sie sie mit Ihrem API-Schlüssel konfigurieren. Setzen Sie dazu die Umgebungsvariable `OPENAI_API_KEY` auf den Wert Ihres API-Schlüssels, den Sie im Dashboard Ihres OpenAI-Kontos finden.

Hyperparameter auswählen und optimieren

Nachdem wir unser JSONL-Dokument erstellt und die OpenAI-CLI installiert haben, können wir nun unsere Hyperparameter auswählen. Die folgende Liste gibt die wichtigsten Hyperparameter mit ihren Definitionen an:

- Die *Lernrate* bestimmt die Größe der Schritte, die das Modell während der Optimierung unternimmt. Bei einer kleineren Lernrate konvergiert das Training langsamer, liefert aber möglicherweise ein genaueres Modell, während eine größere Lernrate das Training beschleunigt, dabei aber über die optimale Lösung hinausschießen kann.
- Die *Stapelgröße* bezieht sich auf die Anzahl der Trainingsbeispiele, die in einer einzelnen Iteration der Modellaktualisierung verwendet wird. Mit größeren Stapeln läuft das Training schneller ab, und die Gradienten sind in der Regel stabiler, während eine kleinere Stapelgröße zu einem genaueren Modell führen kann, allerdings auf Kosten einer langsameren Konvergenz.
- Eine *Trainingsepoche* ist ein vollständiger Durchlauf durch das gesamte Trainingsdatenset. Die Anzahl der Trainingsepochen bestimmt, wie oft das Modell über die Daten iteriert, um zu lernen und seine Parameter feinzutunen.

OpenAI hat viel Arbeit investiert, um für die meisten Fälle optimale Einstellungen zu finden. Daher werden wir uns bei unserem ersten Versuch auf die Empfehlungen von OpenAI verlassen. Als einzige Änderung reduzieren wir das Training von den standardmäßigen vier Epochen auf eine Epoche. Wir wollen nämlich erst einmal sehen, wie die Performance aussieht, bevor wir zu viel Zeit und Geld investieren. Mit verschiedenen Werten zu experimentieren und Techniken wie die Rastersuche zu verwenden, hilft dabei, die optimalen Hyperparametereinstellungen für die jeweilige Aufgabe und das Datenset zu

finden. Bedenken Sie aber, dass dieser Prozess zeitaufwendig und kostspielig sein kann.

Unser erstes feingetuntes LLM

Beginnen wir mit unserem ersten Feintuning. Der Code in Beispiel 4-2 ruft OpenAI auf, um ein Ada-Modell (schnellstes, billigstes, schwächstes) für eine Epoche mit unseren Trainings- und Validierungsdaten zu trainieren.

Beispiel 4-2: Unseren ersten Aufruf für das Feintuning ausführen

```
# Den Befehl 'fine_tunes.create' über die OpenAI-API ausführen
```

```
!openai api fine_tunes.create \
```

```
# Die Datei für das Trainingsdatenset im JSONL-Format  
spezifizieren
```

```
-t "amazon-english-full-train-sentiment.jsonl" \
```

```
# Die Datei für das Validierungsdatenset im JSONL-Format  
spezifizieren
```

```
-v "amazon-english-full-val-sentiment.jsonl" \
```

```
# Berechnung der Klassifizierungsmetriken nach dem  
Feintuning aktivieren
```

```
-compute_classification_metrics \
```

```
# Die Anzahl der Klassen für die Klassifizierung festlegen  
(hier 5)
```

```
-classification_n_classes 5 \
```

```
# Das zu optimierende Basismodell festlegen (hier das  
kleinste Modell, ada)
```

```
-m ada \
```

```
# Die Anzahl der Epochen für das Training festlegen (hier 1)
```

```
-n_epochs 1
```

Feingetunte Modelle mit quantitativen Metriken bewerten

Die Performance von feingetunten Modellen zu messen, ist unabdingbar, um die Effektivität der Modelle zu verstehen und verbesserungswürdige Bereiche zu identifizieren. Metriken und Benchmarks wie Genauigkeit, F1-Maß oder Komplexität bieten quantitative Maße für die Performance des Modells. Neben quantitativen Metriken können auch qualitative Bewertungstechniken wie die Bewertung und Analyse von Beispielausgaben durch den Menschen wertvolle Einblicke in die Stärken und Schwächen des Modells bieten und helfen, Bereiche zu identifizieren, die reif für ein weiteres Feintuning sind.

Nach einer Epoche (Abbildung 4-6 zeigt weitere Metriken) hat unser Klassifizierer eine Genauigkeit von über 63 % mit dem Holdout-Testdatenset erreicht. Es sei daran erinnert, dass das Testdatenset nicht an OpenAI übergeben wurde. Stattdessen haben wir es für die endgültigen Modellvergleiche aufgehoben.

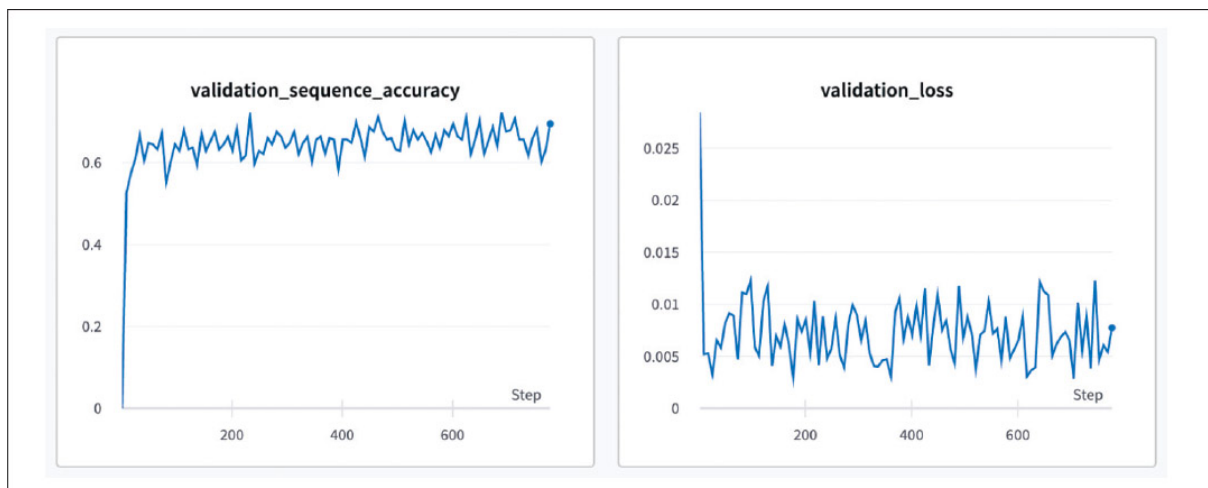


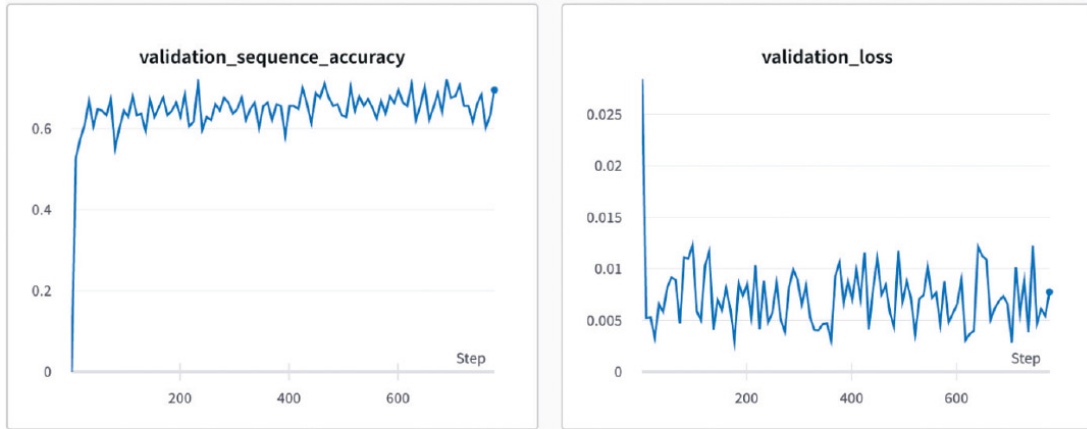
Abbildung 4-6: Unser Modell schneidet nach nur einer Epoche gut ab, wenn es mit gemischten Trainingsdaten und entfernten Duplikaten trainiert wurde.

Eine Genauigkeitsrate von 63 % mag Ihnen niedrig erscheinen, doch sehen Sie es einmal so: Die Vorhersage der genauen Anzahl von Sternen ist schwierig, weil

die Rezensenten nicht nach einheitlichen Richtlinien schreiben und das Produkt entsprechend subjektiv bewerten. Deshalb schlage ich zwei weitere Metriken vor:

- Schwächt man die Genauigkeitsberechnung auf Binärwerte ab (hat das Modell drei oder weniger Sterne vorhergesagt und wurde das Produkt tatsächlich mit drei oder weniger Sternen bewertet), entspricht dies einer Genauigkeitsrate von 92 %, d.h., das Modell kann zwischen »gut« und »schlecht« unterscheiden.
- Schwächt man die Berechnung auf »einmalig« ab, sodass das Modell beispielsweise vorhersagt, dass zwei Sterne als korrekt zählen, wenn die tatsächliche Bewertung einen, zwei oder drei Sterne lautet, entspricht dies einer Genauigkeitsrate von 93 %.

Und wissen Sie was? Das ist gar nicht so schlecht. Unser Klassifizierer lernt zweifellos den Unterschied zwischen gut und schlecht. Der nächste logische Gedanke könnte sein: »Machen wir mit dem Training weiter!« Wir haben nur für eine einzige Epoche trainiert, also müssten mehr Epochen besser sein, oder nicht? Dieser Prozess, in kleineren Schritten zu trainieren und bereits feingetunte Modelle für mehr Trainingsschritte/-epochen mit neuen beschrifteten Datenpunkten zu aktualisieren, wird als *inkrementelles Lernen* bezeichnet, auch als kontinuierliches Lernen oder Online Learning bekannt. Inkrementelles Lernen führt oftmals zu kontrolliertem Lernen, was ideal sein kann, wenn man mit kleineren Datensets arbeitet oder einen Teil des allgemeinen Wissens des Modells bewahren möchte. Probieren wir etwas inkrementelles Lernen aus! Wir nehmen unser bereits feingetuntes Ada-Modell und lassen es drei weitere Epochen mit denselben Daten laufen. Abbildung 4-7 zeigt die Ergebnisse.



Oben: Gemischte Stimmungsdaten im Training ergeben schon nach nur einer Epoche keine schlechten Ergebnisse.

Unten: Wird das Modell inkrementell weitere drei Epochen trainiert, sind keine signifikanten Änderungen festzustellen.

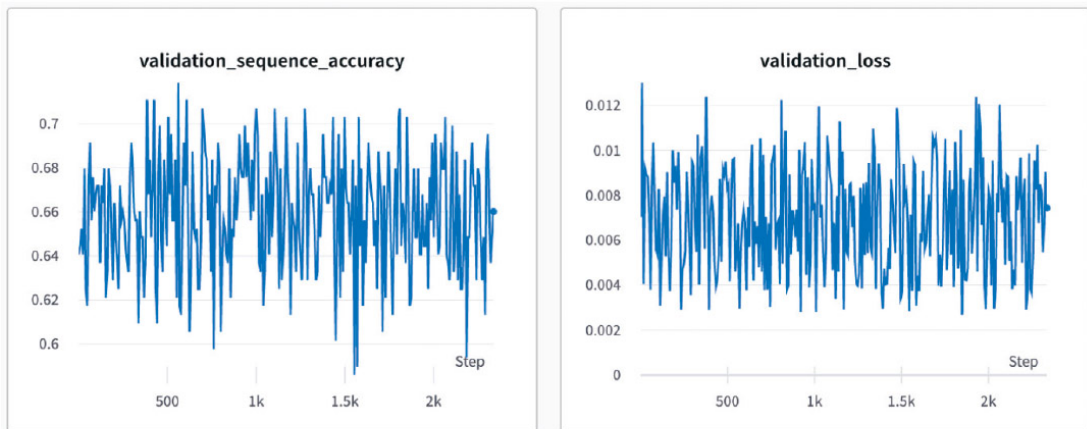


Abbildung 4-7: Bereits nach einer erfolgreichen Epoche scheint sich die Performance des Modells durch inkrementelles Lernen während weiterer drei Epochen kaum zu verändern. Die vierfachen Kosten für die 1,02-fache Leistung? Nein, danke.

Oha! Mehr Epochen scheinen nicht wirklich etwas zu bewirken. Aber nichts ist in Stein gemeißelt, solange wir nicht mit unserem Testdatenset testen und die Ergebnisse mit unserem ersten Modell vergleichen. Tabelle 4-1 zeigt die Ergebnisse.

Tabelle 4-1: Ergebnisse

Quantitative Metrik (mit Testdaten, falls anwendbar)	1 Epoche Stimmungs- klassifizierer: ungemischte Daten	1 Epoche Stimmungs- klassifizierer: gemischte Daten	4 Epochen Stimmungs- klassifizierer: gemischte Daten
Genauigkeit	32 %	63 %	64 %
»Gut« vs. »Schlecht«	70 %	92 %	92 %
Einmalige Genauigkeit	71 %	93 %	93 %
Kosten des Feintunings (gesamt in US-Dollar)	\$ 4,42	\$ 4,42	\$ 17,68

Für den vierfachen Preis bekommen wir also nur einen einzigen Prozentpunkt an Genauigkeit? Meiner Meinung nach ist das den Aufwand nicht wert, aber vielleicht sieht es bei Ihnen anders aus? In einigen Branchen müssen die Modelle nahezu perfekt sein, und da kommt es schon auf einzelne Prozentpunkte an. Die Entscheidung überlasse ich Ihnen, weise aber darauf hin, dass mehr Epochen nicht immer zu besseren Ergebnissen führen. Inkrementelles oder Online Learning kann Ihnen dabei helfen, den richtigen Punkt für das Ende des Trainings zu finden. Dies ist zwar mit höherem Aufwand verbunden, der sich aber auf lange Sicht auszahlt.

Qualitative Bewertungstechniken

Neben quantitativen Metriken bieten qualitative Bewertungstechniken wertvolle Einblicke in die Stärken und Schwächen eines feingetunten Modells. Untersucht man die generierten Ausgaben und zieht dabei menschliche Bewerter hinzu, lassen sich Bereiche ausmachen, in denen das Modell überragend oder unzureichend ist. Daran können wir uns für zukünftiges Feintuning orientieren.

Zum Beispiel können wir die Wahrscheinlichkeit für unsere Klassifizierung ermitteln, indem wir die Wahrscheinlichkeiten für die Vorhersage des ersten Tokens entweder im Playground (wie Abbildung 4-8 zeigt) oder über den Wert `logprobs` der API (wie in Beispiel 4-3 zu sehen) abrufen.

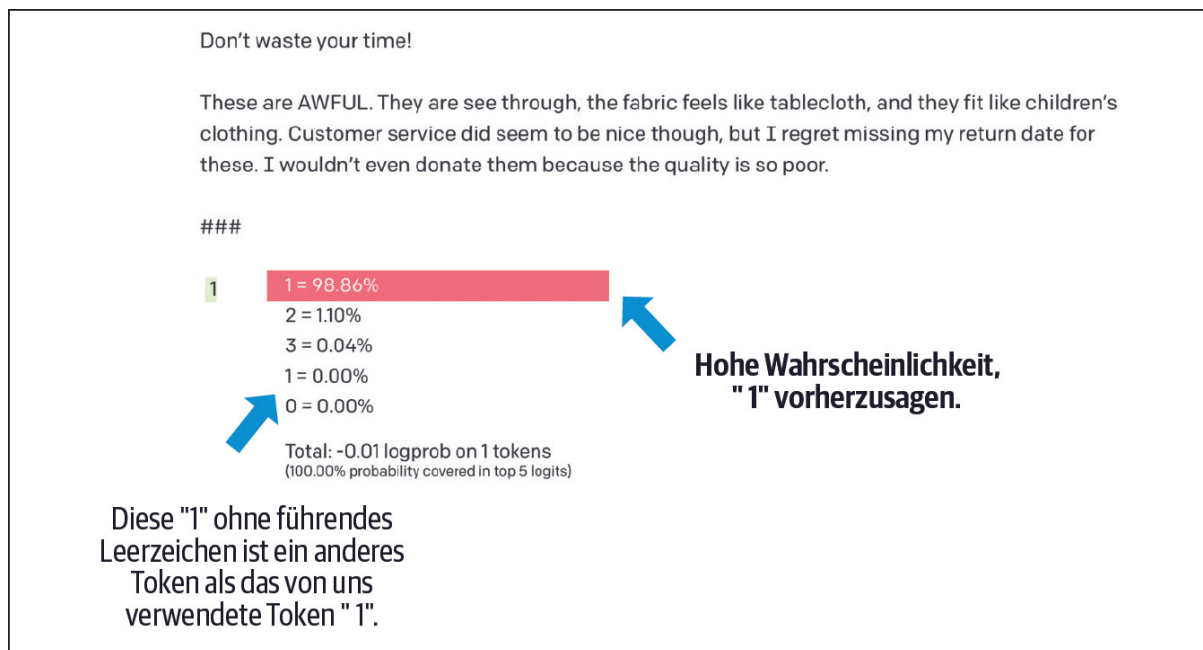


Abbildung 4-8: Der Playground und die API für GPT-3-ähnliche Modelle (einschließlich unseres feingetunten Ada-Modells, wie in dieser Abbildung zu sehen) liefern Token-Wahrscheinlichkeiten, mit denen wir die Vertrauenswürdigkeit des Modells für eine bestimmte Klassifizierung überprüfen können. Beachten Sie, dass die Hauptoption »1« genau wie in unseren Trainingsdaten ein führendes Leerzeichen enthält, aber eines der Token am Anfang der Liste »1« ohne führendes Leerzeichen lautet. Da viele LLMs dies als zwei separate Token betrachten, weise ich so oft auf diese Unterscheidung hin. Es kann leicht passieren, dass man nicht darauf achtet und die Token verwechselt.

Beispiel 4-3: Token-Wahrscheinlichkeiten von der OpenAI-API abrufen

```
import math

# Einen zufälligen Prompt aus dem Testdatenset auswählen

prompt = english_test_df['prompt'].sample(1).iloc[0]

# Eine Completion mit dem feingetunten Modell generieren

res = openai.Completion.create(

    model='ada:ft-personal-2023-03-31-05-30-46',
```

```

    prompt=prompt,

    max_tokens=1,

    temperature=0,

    logprobs=5,

)

# Eine leere Liste initialisieren, um Wahrscheinlichkeiten zu
speichern

probs = []

# logprobs-Werte aus der API-Antwort extrahieren

logprobs = res['choices'][0]['logprobs']['top_logprobs']

# logprobs-Werte in Wahrscheinlichkeiten umrechnen und in der
Liste

# 'probs' speichern

for logprob in logprobs:

    _probs = {}

    for key, value in logprob.items():

        _probs[key] = math.exp(value)

    probs.append(_probs)

```

```

# Die vorhergesagte Kategorie (star) aus der API-Antwort
extrahieren

pred = res['choices'][0].text.strip()

# Den Prompt, die vorhergesagte Kategorie und die
Wahrscheinlichkeiten

# ordentlich ausgeben

print("Prompt: \n", prompt[:200], "... \n")

print("Predicted Star:", pred)

print("Probabilities:")

for prob in probs:

    for key, value in sorted(prob.items(), key=lambda x: x[1],
reverse=True):

        print(f"{key}: {value:.4f}")

    print()

```

Ausgabe:

Prompt:

Great pieces of jewelry for the price

Great pieces of jewelry for the price. The 6mm is perfect for my tragus piercing. I gave four stars because I already lost one because it fell out! Other than that I am very happy with the purchase!

Predicted Star: 4

Probabilities:

4: 0.9831

5: 0.0165

3: 0.0002

2: 0.0001

1: 0.0001

Zwischen quantitativen und qualitativen Maßnahmen nehmen wir an, dass unser Modell für den Einsatz in der Produktion bereit ist – oder zumindest in einer (Ausführungs-)Umgebung für weitere Tests. Überlegen wir aber zunächst einmal, wie wir unser neues Modell in unsere Anwendungen einbinden können.

Feingetunte GPT-3-Modelle in Anwendungen integrieren

Ein feingetuntes GPT-3-Modell in eine Anwendung zu integrieren, ist identisch damit, ein von OpenAI bereitgestelltes Basismodell zu verwenden. Beide Varianten unterscheiden sich nur dadurch, dass Sie auf den eindeutigen Bezeichner Ihres feingetunten Modells verweisen müssen, wenn Sie API-Aufrufe durchführen. Hier sind die wichtigsten Schritte, die Sie befolgen müssen:

- 1. Identifizieren Sie Ihr feingetuntes Modell:** Nachdem das Feintuning abgeschlossen ist, erhalten Sie einen eindeutigen Bezeichner für Ihr feingetuntes Modell – etwas in der Art wie `'ada:ft-personal-2023-03-31-05-30-46'`. Notieren Sie sich diesen Bezeichner, da er für API-Aufrufe erforderlich ist.
- 2. Verwenden Sie die OpenAI-API normal:** Fragen Sie Ihr feingetuntes Modell über die OpenAI-API ab. Ersetzen Sie in den Anfragen den Namen des Basismodells durch den eindeutigen Bezeichner Ihres feingetunten Modells. Beispiel 4-3 gibt hierfür ein Beispiel an.

3. **Passen Sie die Anwendungslogik an:** Da feingetunte Modelle möglicherweise andere Prompt-Strukturen erfordern oder andere Ausgabeformate erzeugen, müssen Sie die Logik Ihrer Anwendung aktualisieren, um mit diesen Variationen umgehen zu können. Zum Beispiel haben wir in unseren Prompts den Titel mit dem Text der Rezension verkettet und ein benutzerdefiniertes Suffix "\n\n#\n\n#" angefügt.
4. **Überwachen und bewerten Sie die Performance:** Die Performance Ihres Modells sollten Sie kontinuierlich überwachen und Feedback vom Benutzer einholen. Gegebenenfalls müssen Sie Ihr Modell iterativ mit noch mehr Daten feintunen, um seine Genauigkeit und Effektivität zu verbessern.

Fallstudie 2: Klassifizierung der Kategorien von Amazon-Rezensionen

Nachdem wir nun ein Ada-Modell erfolgreich für ein relativ einfaches Beispiel wie Stimmungsklassifizierung feingetunt haben, wollen wir uns an eine anspruchsvollere Aufgabe wagen.

In einer zweiten Fallstudie untersuchen wir, wie das Feintuning eines GPT-3-Modells dessen Performance verbessern kann, die Kategorien von Amazon-Rezensionen aus demselben Datenset zu klassifizieren. Bei dieser Aufgabe geht es auch um die Klassifizierung der Amazon-Rezensionen in die jeweiligen Produktkategorien basierend auf dem Titel und dem Text der Rezension, so wie wir es bei der Stimmungsanalyse gemacht haben. Allerdings haben wir zum Beispiel nicht mehr fünf Klassen, sondern stattdessen 31 unausgewogene Klassen (siehe Abbildung 4-9).

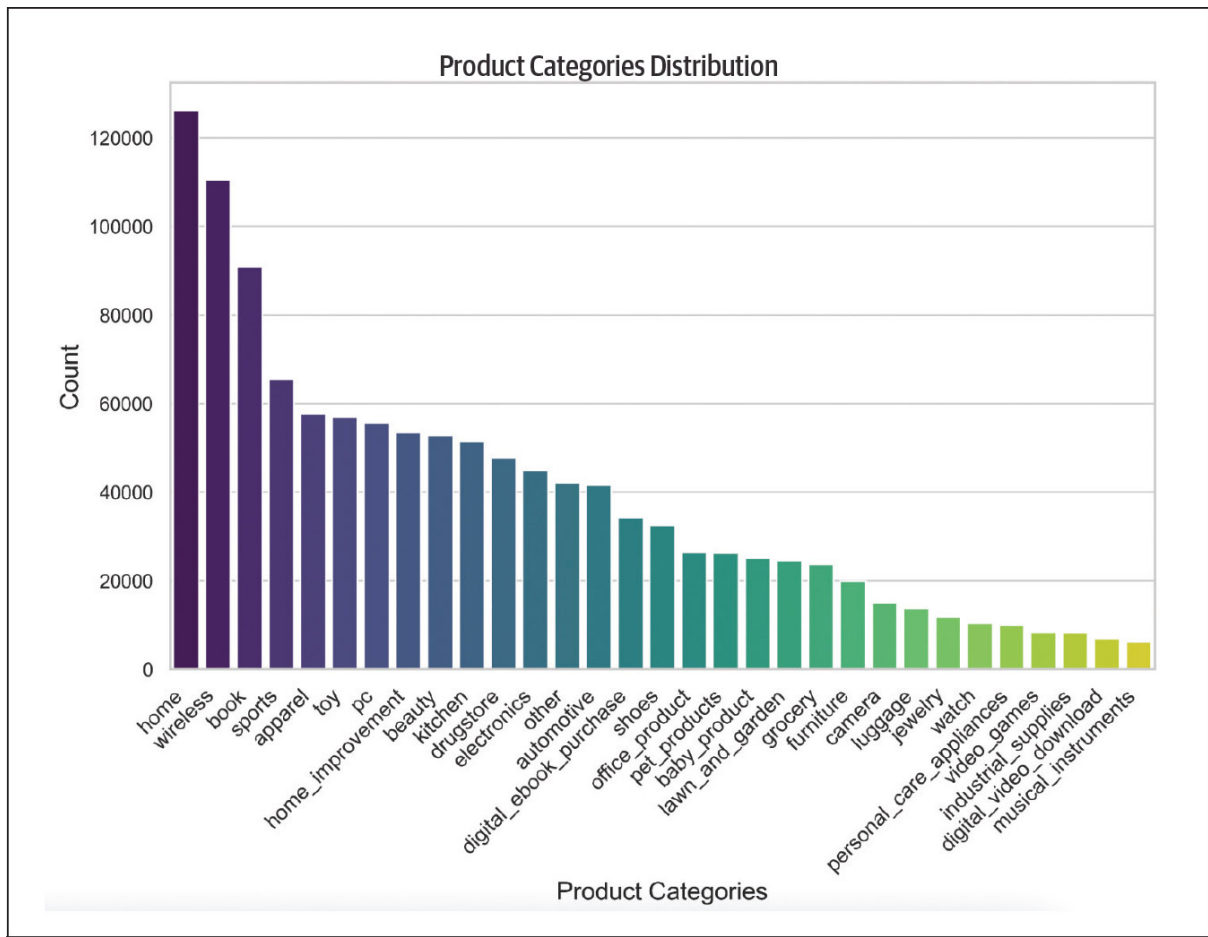


Abbildung 4-9: Bei der Klassifizierungsaufgabe stehen 31 eindeutige Kategorien zur Auswahl, wobei die Verteilung der Klassen sehr unausgewogen ist. Dies ist ein perfektes Gedränge, das eine schwierige Klassifizierungsaufgabe schafft.

Die bedeutend schwierigere Aufgabe der Klassifizierung von Kategorien offenbart viele versteckte Schwierigkeiten, die mit Machine Learning verbunden sind, beispielsweise den Umgang mit *unausgewogenen* und *schlecht definierten Daten* – wobei die Unterscheidung zwischen den Kategorien subtil oder mehrdeutig ist. In diesen Fällen hat das Modell Mühe, die korrekte Kategorie zu erkennen. Um die Performance zu verbessern, sollten Sie die Problemdefinition verfeinern, redundante oder verwirrende Trainingsbeispiele löschen, ähnliche Kategorien zusammenführen oder dem Modell über Prompts zusätzlichen Kontext anbieten. Im Code-Repository zum Buch können Sie alle diese Arbeiten nachlesen (<https://github.com/sinanuozdemir/quickstart-guide-to-llms>).

Zusammenfassung

Das Feintunen von LLMs wie GPT-3 ist eine effektive Methode, um die Performance bei bestimmten Aufgaben oder in bestimmten Bereichen zu verbessern. Wenn Sie ein feingetuntes Modell in Ihre Anwendung integrieren

und sich an die Empfehlungen für die Bereitstellung halten, können Sie eine effizientere, genauere und kostengünstigere Lösung für die Sprachverarbeitung realisieren. Indem Sie die Performance Ihres Modells kontinuierlich überwachen und bewerten sowie das Modell immer wieder optimieren, stellen Sie sicher, dass es die steigenden Anforderungen Ihrer Anwendung und Ihrer Benutzer erfüllen kann.

Auf das Konzept des Feintunings kommen wir in späteren Kapiteln mit einigen komplexeren Beispielen zurück. Dann untersuchen wir auch Strategien des Feintunings für Open-Source-Modelle, um die Kosten noch weiter zu senken.

A

Ähnlichkeit

Jaccard-Koeffizient 159

Kosinus- 58, 152

Alignment 78

Ausrichtung 259

all-mpnet-base-v2 165, 166

alpha 236

Amazon, Rezensionen 116

Angriffe, Prompt Injection 119

ANNOY 68

Apache 2.0 244

API 70

Closed Source 251

API-Aufrufe 231

Architekturen, Transformer 27

Attention 37, 257, 258

AUC (Area Under the Curve) 199

Auffüllen, DataCollatorWithPadding 204

Aufmerksamkeit 37, 258

Ausrichtung 78, 259

AutoModelForCausalLM 214
AutoModelForSequenceClassification 214
AWS CloudWatch 254

B

BART 25
Batch Size 261
BERT (Bidirectional Encoder Representations from) 25, 42, 258
Beschneiden, LLM 234
Beschreibungsfelder 157
bestärkendes Lernen *siehe* Reinforcement Learning
Bewertungsmetriken 261
Bibliotheken
 LIME 255
 Optuna 254
 Pydantic 70
 Sentence Transformers 60, 61, 73, 163
 transformers 191

Bi-Encoder 60

BioGPT 45

BM25 69

BoolQ 72

C

cased 40

Casing 40

Chain-of-Thought (CoT) *siehe* Gedankenketten

Chatbot 89

ChatGPT 42, 77, 85, 89

Chunking 62

CLI (Command Line Interface) 106, 110

- OpenAI 110
- Closed Source
 - API 251
 - Kosten 75
- Clustering, semantische Dokumente 65
- Cohere 86
- compile 206
- Cross-Attention 176
- Cross-Encoder 69

D

- Data Collator 200
- DataCollatorWithPadding 204
- Daten, beschriftete 260
- Datenbanken
 - ANNOY 68
 - Pgvector 68
 - Pinecone 68
 - Vektor- 68
 - Weaviate 68
- Datensets
 - BoolQ 72
 - Englisch – LaTeX 212
 - Holdout 102
 - MNLI (Multi-Genre Natural Language Inference) 122
 - Open Instruction Generalist (OIG) 217
 - Visual QA 182
- Decoder 257
- Deduplizierung semantischer Ähnlichkeiten 203
- Destillation

- aufgabenabhängige 234
- aufgabenunabhängige 234
- DistilBERT 173
- Modelle 173
- RoBERTa 159

Detraktor 156

DistilBERT 173

DistillationTrainer 236

Dokumente

- Chunking 62
- ohne Chunking 67
- semantische 65

Drop-out 233, 253

E

Einfrieren, Modellgewichte 207

Embedder, Basis- 159

Embeddings 39, 58

- Chunking 62

- Engines 58, 59

Encoder 257

Engines 59

eval() 232

Exploration 151, 167

F

F1-Maß 199

Facebook

- BART 25

- RoBERTa 34

Fakten, erfundene Gedankenketten 131

FAQs (Frequently Asked Questions) 251

FastAPI 70

Feintuning 36, 101, 260

 Schleife 200

 Transfer Learning 100

Few-Shot-Learning 81

Fläche unter der Kurve 199

forward 179

Frage-Antwort-Bot mit ChatGPT 89

Frage-Antwort-System, visuelles 171

Funktionen

 get_embeddings 60

 save_pretrained() 243

 softmax 238

Fusion 176

G

Gedankenketten 134

 testen 138

 Testen auf Prompt Injection 253

Genre

 Vorhersagemodell 233

 Vorhersagen 198, 239

get_embeddings 60

GNU GPL v3 244

Google, BERT 25

GPT (Generative Pre-trained Transformers)

 ChatGPT 43, 77

 GPT-3 43, 104

GPT-4 77, 89

Grafana 254
Groß-/Kleinschreibung 40
große Sprachmodelle 258
Grundkenntnisse, dem LLM hinzufügen 251

H

Halluzinationen 131
Holdout 102
Hugging Face 243

- CLI 245
- Trainer 200
- TrainingArguments 200

huggingface_hub 245
Hyperparameter 261

- alpha 236
- auswählen 110
- Lernrate 110
- optimieren 110
- Optuna 254
- Stapelgröße 110
- temperature 236
- Trainingsepochen 111

I

Inference Endpoints, Huggins Face 246
Infrastruktur 169
init 179
inkrementelles Lernen 112, 261
InstructGPT 222

J

Jaccard-Koeffizient 153, 198

K

Klassen

AutoModelForCausalLM 214

AutoModelForSequenceClassification 214

Klassifizierung, Multilabel 198

KL-Divergenz 193

Kompatibilität 233

Konstruktoren 179

Korpora 260

Kosinus-Ähnlichkeit 58, 152

Kosten

API-Aufrufe 231

Closed Source 75

LLMs 243

Rechnerkosten 243

Kreuzentropie 187

Kullback-Leibler-Divergenz 236

L

L2-Regularisierung 253

Labeled Data 260

LangChain 253

Large Language Models (LLMs) 258

all-mpnet-base-v2 165

Archetypen 263

beschneiden 234

Definition 28

dialogorientierte 89

domänenspezifische 45

Funktionsweise 33

Größe reduzieren 234

Hauptmerkmale 30

mehrsprachige 254

Überanpassung 262

Unteranpassung 262

Verlustfunktion 187

LaTeX 211

Lernen

bestärkendes (Reinforcement Learning) 186

inkrementelles 112, 261

online 112, 261

Lernrate 110, 261

Levenshtein-Distanz 130

LIME 255

Lizenzen 244

Logit 190

M

Magic: The Gathering 53

mBERT (multilingual BERT) 254

Methoden

forward 179

init 179

Konstruktoren 179

Metriken 261

MIT 244

Mixed-Precision Training 205

MNLI (Multi-Genre Natural Language Inference) 122

Modelle

all-mpnet-base-v2 166

BERT 42
Cohere 86
DistilBERT 173
Einfrieren der Gewichte 207
GPT 43
Hugging Face 243
Lernrate 261
paraphrase-distilroberta-base-v1 166
T5 44
Überanpassung 262
Unteranpassung 262

Modellkarte 244

Musterexploitation 151

N

Natural Language Processing (NLP) 258

negativer Log-Likelihood-Verlust 220

n-Gram 28

NLP (Natural Language Processing) 258

O

Online Learning 112, 261

ONNX (Open Neural Network Exchange) 233

OpenAI

 Befehlszeilenschnittstelle 110

 CLI 106, 110

 Embeddings 58

 Engines 59

optimum 233

Optuna 254

P

Pakete

- huggingface_hub 245
- optimum 233
- pip 110

paraphrase-distilroberta-base-v1 166

PEFT LoRA 228

Personas 83

Pgvector 68

Pinecone 68

PPO (Proximal Policy Optimization) 188, 191

Produktion, Closed Source 231

Projektion 176

Promotor 156

Prompt Chaining 126

Prompt Engineering 77, 259

- Open-Source-Modelle 212

Prompt Injection 119, 252

- Verkettung 129

Prompts 259

- ChatGPT 85
- Gedankenketten 134
- modellübergreifend 85
- Stapelverarbeitung 125
- System- 85
- Verkettung 126

Prompt Stuffing 130

Proximal Policy Optimization (PPO) 188, 191

push_to_hub 245

Pydantic 70

Python

FastAPI 70

LIME 255

Q

Question-Answering *siehe* Frage-Antwort

R

README.md 245

Rechnerkosten 243

Reinforcement Learning 186

Bibliotheken 191

durch menschliche Rückkopplung 42, 259

Feedback 186

from AI Feedback (RLAIF) 260

from Feedback (RLF) 187

mit Rückkopplung 187, 259

Proximal Policy Optimization (PPO) 188

Trust Region Policy Optimization (TRPO) 188

Rezensionen, klassifizieren 116

RLAIF (Reinforcement Learning from AI Feedback) 260

RLF (Reinforcement Learning from Feedback) 187

RLHF (Reinforcement Learning from Human Feedback (RLHF, durch menschliche Rückkopplung) 42, 259

RoBERTa 34

ROC (Receiver Operating Characteristic) 199

ROC/AUC 199

S

Sammelkartenspiel *siehe* Magic:

The Gathering

save_pretrained() 243

SAWYER, Sinan's Attempt at Wise Yet Engaging Responses 215

Schichten

deaktivieren 233

Drop-out 233

Schreibweise

Casing 40

Umlaute 40

Selbstaufmerksamkeit 27, 173

Self-Attention 27, 173

sentence_transformers 61, 73

Sentence Transformers 60, 163

Sentiment 189

softmax 238

sokratische Methode 137

Sprachmodelle

Ausrichtung 78

autoencodierende 29, 258

autoregressive 29, 258

Stapelgröße 110, 261

Star Wars 94

Strukturierung, Ausgabe 82

Suche, semantische 55, 71

Systemprompts 85

T

T5 44

temperature 236, 238

TensorBoard 254

Token 28, 258

Tokenisierung 40

Casing 40

Tools

AWS CloudWatch 254

Drittanbieter 253

Grafana 254

LangChain 253

TensorBoard 254

`torch.compile(model)` 206

Trainer 200

DistillationTrainer 236

`trainer.create_model_card()` 245

Training

Hyperparameter 261

Mixed-Precision 205

TrainingArguments 200

Trainingsepochen 111, 261

Transfer Learning 35, 100, 259

Feintuning 101

Transformer 27

Architektur 257

DataCollatorWithPadding 204

Modell übertragen 245

Vision Transformer (ViT) 173

transformers, Bibliothek 191

TRPO (Trust Region Policy Optimization) 188

Trust Region Policy Optimization (TRPO) 188

Türkisch 40

U

Überanpassung 262

Drop-out 253
L2-Regularisierung 253
Übersetzungen, maschinelle 47
Umlaute 40
uncased 40
Unteranpassung 253, 262
uvicorn 70

V

Vektordatenbanken 68
Verkettung 129
 Prompts 126
Verluste
 Kreuzentropie 187
 Kullback-Leibler-Divergenz 236
Verlustfunktionen
 differenzierbar 220
 eigene 220
 InstructGPT 222
 negativer Log-Likelihood-Verlust 220
 temperature-Quadrat 238
Vision Transformer (ViT) 173
Visual QA 182
visuelles Frage-Antwort-System 171
ViT (Vision Transformer) 173
Vorhersagemodell, Genre 233
Vorhersagen, Genre 239
Vortraining 33
VQA, Trainingsschleife 183

W

Weaviate 68

WebCrawl 211

Wissensdestillation 173, 234

temperature 238

X

XLM (Cross-lingual Language Model) 254