AI Business Cases mit SAP

Szenarien, Tools und Best Practices

Kapitel 4

Anomalieerkennung in Finanztransaktionen

Deep Learning und Machine Learning erkennen in Finanzdaten potenziell betrügerische oder fehlerhafte Transaktionen. Der Einsatz des SAP AI Core erlaubt die Echtzeitanalyse großer Datenmengen, was Risiken reduziert und Compliance verbessert.

Nachdem wir nun schon einige Anwendungsfälle in der Logistik besprochen haben, wenden wir uns in diesem Kapitel einem Anwendungsfall aus dem Finanzwesen zu.

Die Erkennung von Auffälligkeiten in Finanzdaten ist eine zentrale Herausforderung im digitalen Zeitalter. Gerade aufgrund großer Datenmengen und zunehmend automatisierter Prozesse ist es wichtig, geeignete Methoden für die zuverlässige Detektion von Abweichungen zu kennen und praktisch einzusetzen. Hierbei spielen die Auswahl und Vorbereitung der Daten ebenso eine Rolle wie die Definition, was im jeweiligen Kontext als »normal« oder »anomal« gilt. Aufbauend darauf lernen Sie, wie sich geeignete Algorithmen – exemplarisch der Isolation Forest – für die Anomalieerkennung implementieren lassen.

Neben der technischen Umsetzung rückt auch die Einbettung der Lösung in bestehende IT-Landschaften in den Fokus. Sie sehen, wie Sie APIs entwickeln, Trainings- und Deploymentszenarien mit Docker und ArgoFlows in SAP AI Core automatisieren und KI-Services mit KServe produktiv bereitstellen. Abschließend wird die Integration und Bereitstellung der KI-Lösung auf der SAP Business Technology Platform (SAP BTP) praxisnah erläutert.

Welche Voraussetzungen für die Entwicklung einer zuverlässigen Anomalieerkennung geschaffen werden müssen, zeigt Abschnitt 4.2, »Die richtigen Voraussetzungen schaffen«. Die technischen Details der Umsetzung, von der Algorithmus-Implementierung bis zur Bereitstellung als Service, werden in Abschnitt 4.3, »Technische Umsetzung«, behandelt. Wie Sie die Lösung auf der SAP BTP integrieren und betreiben, erläutert Abschnitt 4.4, »Die KI-Lösung auf der SAP BTP bereitstellen«. Schritt für Schritt erhalten

Sie so einen praxisorientierten Leitfaden, um Anomalieerkennung in Ihren Finanzprozessen erfolgreich einzusetzen.

4.1 Einleitung und Zielsetzung

Ausgangssituation

Data Scientists bei der fiktiven SecureBank AG, einer mittelgroßen europäischen Bank, die ein modernes und breites Portfolio aus Online-Banking, Kreditkarten und Mobile Payment anbietet, beginnen ihren Arbeitstag routinemäßig mit einer genauen Prüfung der aktuellen Transaktionsübersicht auf dem Dashboard für Betrugserkennung. Hinter diesem Dashboard steht ein komplexes technisches System, das täglich mehrere Tausend Finanztransaktionen in Echtzeit analysiert und überwacht.

Gewöhnliche und legitime Transaktionen dominieren typischerweise das Bild: Kundinnen und Kunden kaufen Lebensmittel im örtlichen Supermarkt, tanken ihre Fahrzeuge, besuchen Restaurants zur Mittagszeit oder erwerben abends online Kleidung und Unterhaltungselektronik. Die Beträge solcher alltäglichen Zahlungen liegen meist zwischen 10 und 100 Euro und erfolgen während der üblichen Geschäftszeiten – insbesondere während der Mittagszeit oder am frühen Abend. Die Händlerkategorien sind vertraut und umfassen Lebensmittelgeschäfte, Tankstellen, Bekleidungsgeschäfte sowie Elektronikfachhändler.

Auffällige Muster

Doch an diesem Morgen fällt den Expertinnen und Experten auf, dass sich einige Transaktionen ungewöhnlich verhalten: Um 2:43 Uhr nachts wird zunächst eine extrem niedrige Zahlung von nur 19 Cent bei einer Tankstelle registriert. Solche kleinen Beträge könnten Testtransaktionen sein, bei denen Betrügerinnen und Betrüger prüfen, ob gestohlene oder kompromittierte Karten funktionieren. Nur eine Stunde später folgt eine auffällig hohe Abbuchung von über 3.000 Euro beim selben Händler, allerdings diesmal aus einer Kleinstadt, in der üblicherweise kaum Transaktionen stattfinden. Noch später in der Nacht, um etwa 4:30 Uhr, erfolgt erneut eine winzige Transaktion von lediglich 30 Cent in einem Restaurant. Kurze Zeit darauf wird eine Zahlung über 4.500 Euro für eine Reisebuchung festgestellt. Ein Betrag, der deutlich außerhalb des üblichen Transaktionsrahmens der betroffenen Kundinnen und Kunden liegt.

Nicht der Händler, sondern das Muster ist relevant

Diese Kombinationen von ungewöhnlich kleinen Testbeträgen, gefolgt von großen Transaktionen an Orten und zu Zeiten, die nicht zum üblichen Kundenverhalten passen, bilden ein typisches Muster für Betrugsversuche. Dabei sind nicht die Händlerkategorien selbst verdächtig, denn Tankstellen, Restaurants und Reisebüros sind etablierte und legitime Zahlungsempfän-

ger, sondern vielmehr die ungewöhnlichen Kombinationen aus Uhrzeit, Standort und Betrag, die die Transaktionen verdächtig machen.

Um betrügerische Aktivitäten künftig noch schneller und zuverlässiger erkennen zu können, plant die SecureBank AG den Einsatz moderner KI-gestützter Methoden. Derzeit werden verschiedene Machine-Learning-Algorithmen – insbesondere spezialisierte Verfahren wie Isolation Forests oder neuronale Netze – evaluiert, um eine automatisierte Echtzeiterkennung von Anomalien in Finanztransaktionen zu ermöglichen. Ziel ist es, mithilfe von KI-Modellen aus historischen Daten typische Muster und Abweichungen zu identifizieren und verdächtige Transaktionen unmittelbar an das Fraud-Prevention-Team der Bank zu melden.

Dabei ist den Verantwortlichen bewusst, dass nicht jede ungewöhnliche Transaktion zwangsläufig auf Betrug hindeutet. Beispielsweise kann auch eine spontane Tankaktion einer jungen Kundin oder eines Kunden nach einer nächtlichen Party eine Auffälligkeit im System erzeugen, ohne dass ein Betrugsfall vorliegt. Umso wichtiger ist es, die Algorithmen sorgfältig zu trainieren und das Zusammenspiel zwischen automatisierter Erkennung und manueller Überprüfung optimal zu gestalten. Die frühzeitige Identifikation echter Betrugsversuche soll dazu beitragen, finanzielle Schäden zu vermeiden und den Aufwand für Rückerstattungen und Schadenregulierung zu minimieren.

Die Einführung einer automatisierten und KI-gestützten Anomalieerkennung soll somit das Vertrauen der Kundinnen und Kunden in die SecureBank AG stärken, finanzielle Risiken reduzieren und gleichzeitig sicherstellen, dass regulatorische Anforderungen an Sicherheit und Compliance umfassend erfüllt werden.

Vielfältige Anwendungsmöglichkeiten

Doch die angestrebte Lösung bietet Potenzial weit über die reine Betrugserkennung bei Finanztransaktionen hinaus. Künftig sollen auch weitere Anwendungsfelder geprüft werden: beispielsweise die Überwachung interner Buchungen in SAP-Buchungsjournalen oder die automatisierte Detektion ungewöhnlicher Materialbewegungen im Bereich Lager- und Materialwirtschaft. So kann die SecureBank AG von der Anomalieerkennung in verschiedenen Unternehmensbereichen profitieren und die Lösung sukzessive weiter ausbauen.

Im weiteren Verlauf dieses Kapitels begleiten Sie die SecureBank AG bei der schrittweisen Einführung und technischen Umsetzung der KI-basierten Anomalieerkennung – von den notwendigen Voraussetzungen über die Implementierung bis hin zur produktiven Bereitstellung auf der SAP Business Technology Platform.

4.2 Die richtigen Voraussetzungen schaffen

Klarheit von Anfang an

Die automatische Erkennung von Betrugsfällen oder fehlerhaften Transaktionen mithilfe von KI-Verfahren erfordert ein systematisches und methodisches Vorgehen. Entscheidend für den Erfolg eines solchen Projekts ist, von Beginn an klar zu definieren, welche Art von Anomalien erkannt werden soll und wie diese sich vom normalen Verhalten abgrenzen lässt. Ebenso wichtig ist es, die erforderlichen Voraussetzungen zu schaffen, wie z. B. die Verfügbarkeit qualitativ hochwertiger Daten, geeignete technische Infrastruktur sowie die Auswahl und Integration passender KI-Methoden in bestehende Geschäftsprozesse.

Zusammenarbeit ist Pflicht

Ein systematischer Ansatz stellt sicher, dass die Anomalieerkennung nicht nur technisch einwandfrei funktioniert, sondern auch organisatorisch und regulatorisch tragfähig umgesetzt werden kann. Dabei ist es unerlässlich, dass Fachbereich, IT und Compliance von Anfang an eng zusammenarbeiten. Klar definierte Prozesse, Zuständigkeiten und Schnittstellen zwischen den beteiligten Teams helfen dabei, erkannte Anomalien schnell und effizient zu bearbeiten sowie Maßnahmen rechtzeitig einzuleiten.

Die drei Erfolgsfaktoren

Um diese Herausforderung erfolgreich zu meistern, sollten die folgenden drei zentralen Aspekte im Detail betrachtet werden:

- Welche Daten werden benötigt und wie müssen diese vorbereitet sein?
- Was genau ist *normal* und was gilt als *anomal*?
- Welche Algorithmen und KI-Verfahren eignen sich besonders gut zur Erkennung von Anomalien?

Diese drei Kernelemente bilden das Fundament für ein robustes und effizientes Anomalieerkennungssystem, das Betrugsversuche automatisiert und zuverlässig identifiziert, bevor sie Schaden verursachen können. Wir besprechen diese Kernelemente in den folgenden Unterabschnitten dieses Abschnitts.

4.2.1 Daten auswählen und vorbereiten

Relevanz der Datenbasis

Die Qualität und Aussagekraft einer KI-gestützten Anomalieerkennung steht und fällt mit den verfügbaren Daten. Deshalb gilt es zunächst, sorgfältig zu definieren, welche Daten benötigt werden, wie umfangreich diese sein müssen und in welcher Form sie vorliegen sollten. Generell gilt: Je besser und vollständiger die Datenbasis, desto zuverlässiger kann die Anomalieerkennung potenzielle Betrugsfälle identifizieren.

Im Kontext der automatischen Erkennung von Anomalien in Finanztransaktionen kommen typischerweise folgende Datentypen in Betracht:

Datentypen

- **Transaktionsdaten (strukturiert)**

  Transaktionen im Blick

  Die zentrale Datenquelle bilden hier transaktionsbezogene Daten aus Zahlungssystemen oder Buchungsjournalen im SAP-System. Dazu gehören unter anderem:

  – Datum und Uhrzeit der Transaktion

  Beispiel: 25.06.2025, 14:35 Uhr

  – Transaktionsbetrag und Währung

  Beispiel: EUR 54,99

  – Händlerkennung oder Merchant-Kategorie

  Beispiel: Tankstelle, Restaurant, Online-Händler, Reisebüro

  – Zahlungsmethode und Kontotyp

  Beispiel: Visa-Kreditkarte, EC-Karte, Mobile Payment

  – Standortdaten

  Beispiel: Hamburg Innenstadt, Tankstelle München-Süd

  – Kundenidentifikator (anonymisiert)

  Beispiel: Kunden-ID 345987

- **Kundendaten (strukturiert)**

  Kundengruppen verstehen

  Diese Daten erlauben Rückschlüsse auf das Verhalten spezifischer Kundengruppen und umfassen beispielsweise:

  – Demografische Daten

  Beispiel: Alter, Wohnort (Stadt/Land), berufliche Tätigkeit (Student, Rentner, Berufstätige)

  – Kundensegmente oder Verhaltensprofile

  Beispiel: Premiumkunde, Gelegenheitskunde, Intensivnutzer digitaler Kanäle

- **Historische Transaktions- und Verhaltensdaten (zeitlich strukturiert)**

  Vergangenheit analysieren

  Historische Daten sind unerlässlich, um Muster und Abweichungen effektiv zu erkennen. Typische Beispiele sind:

  – Transaktionshistorie der letzten Monate/Jahre

  Beispiel: Kundinnen und Kunden kaufen normalerweise werktags tagsüber zwischen 8 und 20 Uhr.

  – Langzeitprofile einzelner Kundinnen und Kunden

  Beispiel: Eine Kundin bzw. ein Kunde gibt monatlich durchschnittlich ca. 600 Euro aus, jeweils in einem Radius von 50 km um seinen Wohnort.

Technische Spuren nutzen

- **Technische Daten und Metadaten (teilweise unstrukturiert)**

  Darunter fallen weitere technische Parameter, die ergänzende Hinweise geben können:

  – IP-Adressen und Geräteinformationen bei Online-Transaktionen

  Beispiel: Transaktion von einem unbekannten Gerät oder Standort.

  – Verbindungsdaten

  Beispiel: Ungewöhnliche Fehlversuche bei der Anmeldung

Externe Infos einbeziehen

- **Zusatzdaten aus externen Quellen (teilweise unstrukturiert)**

  Zur Ergänzung können externe Daten herangezogen werden, wie etwa:

  – Informationen aus öffentlichen Quellen oder Bonitätsdatenbanken

  Beispiel: Warnungen oder Meldungen zu gestohlenen Kreditkartendaten.

  – Geografische und sozioökonomische Daten

  Beispiel: Regionale Statistiken zu erhöhter Betrugsaktivität.

Daten bereinigen

Um KI-Modelle zur Anomalieerkennung erfolgreich zu trainieren und einzusetzen, ist es unabdingbar, die Rohdaten zunächst sorgfältig aufzubereiten und zu validieren. Dies beginnt mit der *Datenbereinigung*, bei der inkonsistente, fehlerhafte oder unvollständige Datensätze entfernt oder korrigiert werden, um sicherzustellen, dass das KI-Modell auf einer zuverlässigen Datenbasis arbeitet.

Daten normalisieren und skalieren

Danach folgen die *Normalisierung* und die *Skalierung* der Daten: Beträge, Zeitstempel und andere Kennzahlen müssen in eine vergleichbare Form gebracht werden, damit das Modell unterschiedliche Merkmale angemessen miteinander in Verbindung setzen kann.

Wertvolle Merkmale hervorheben

Zudem spielt das sogenannte *Feature Engineering* eine entscheidende Rolle. Dabei werden aus vorhandenen Daten neue, aussagekräftige Merkmale generiert, etwa die Anzahl der Transaktionen einer Kundin oder eines Kunden pro Tag, durchschnittliche Ausgaben je Händlerkategorie oder zeitliche Muster des Zahlungsverhaltens.

Die sorgfältige Auswahl, Aufbereitung und Nutzung geeigneter Daten bilden somit die unverzichtbare Grundlage für eine präzise und zuverlässige KI-gestützte Anomalieerkennung.

Anomalieerkennung in anderen Branchen

Doch nicht nur im Bereich der Finanztransaktionen sind solche Ansätze erfolgsversprechend. In nahezu jeder Branche finden sich vergleichbare Herausforderungen, bei denen Anomalien entscheidende Hinweise auf Risiken oder Probleme liefern können. In der Logistik beispielsweise können

ungewöhnliche Materialbewegungen oder Verzögerungen in Lieferketten auf drohende Engpässe oder Qualitätsprobleme hindeuten. In der Produktion können plötzliche Abweichungen im Maschinenverhalten Vorboten eines bevorstehenden Ausfalls oder einer verringerten Produktqualität sein.

Unabhängig von der Branche und dem konkreten Anwendungsfall gilt jedoch immer: Anomalien existieren überall. Entscheidend ist es, neben dem reinen Verständnis der Daten (*Data Understanding*) auch das Verständnis für den Geschäftskontext (*Business Understanding*) konsequent einzubeziehen. Nur wenn klar ist, was die verfügbaren Daten über das jeweilige Business aussagen, welche Geschäftsprozesse sie abbilden und welche Entscheidungen sie beeinflussen, lassen sich geeignete Anomalien identifizieren, bewerten und entsprechend nutzen, um nachhaltige Mehrwerte zu generieren.

4.2.2 Normal und anomal definieren

Die zentrale Frage bei der Anomalieerkennung ist: Was genau macht eine Transaktion, ein Verhalten oder einen Datenpunkt *normal* oder *anomal*? Diese Unterscheidung klingt zunächst einfach, doch sie ist in der Praxis eine der komplexesten Herausforderungen im Bereich KI-gestützter Analysen.

Definition: Anomalie

Generell gilt: Eine *Anomalie* ist ein Ereignis oder eine Beobachtung, die deutlich von dem abweicht, was allgemein als typisch oder erwartet gilt. Demgegenüber beschreibt »normal« das Verhalten oder Muster, das regelmäßig auftritt und somit vorhersehbar und typisch ist. Um dies zu verdeutlichen, betrachten wir ein einfaches Beispiel.

Beispiel für eine normale Datenreihe

Stellen Sie sich eine Datenreihe vor, die Tankstellen-Umsätze einer Kundin über mehrere Wochen hinweg zeigt:

- Montag, 30.06.: 55 €
- Dienstag, 15.07.: 47 €
- Montag, 28.07.: 60 €
- Mittwoch, 13.08.: 52 €
- Donnerstag, 28.08.: 48 €

Dieses Muster wiederholt sich regelmäßig und spiegelt typische Tankgewohnheiten wider.

Beispiel für eine Anomalie

Plötzlich taucht an einem Mittwoch um 03 Uhr nachts eine Transaktion von 3.000 € bei derselben Tankstelle auf. Eindeutig ungewöhnlich in Bezug auf Betrag, Zeitpunkt und übliche Tankgewohnheiten. Genau diese Abweichung kennzeichnet hier eine Anomalie.

Noise statt Anomalie

Doch nicht jede Auffälligkeit ist zwangsläufig eine Anomalie: Der Begriff *Noise* beschreibt zufällige Schwankungen in Daten, die zwar auffällig erscheinen, aber keine echte Bedeutung oder keinen geschäftlichen Hintergrund haben.

Beispiel für Noise

Ein Kunde, der regelmäßig 40 bis 60 € an der Tankstelle ausgibt, tankt einmalig für 74 €. Etwas höher als gewöhnlich, aber noch innerhalb plausibler Schwankungen. Obwohl leicht auffällig, stellt dies eher Noise als eine echte Anomalie dar.

Anomalien vom Rauschen unterscheiden

Damit KI-Modelle effizient arbeiten, ist es entscheidend, echte Anomalien von einfachem Rauschen zu unterscheiden. Denn, während echte Anomalien häufig geschäftliche Risiken signalisieren, sind Noise-Daten zufällige Ereignisse, die nicht weiter untersucht werden müssen (siehe Abbildung 4.1).

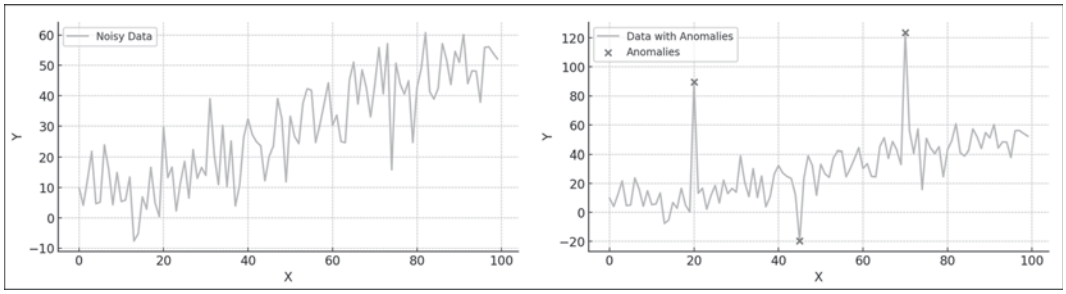

Abbildung 4.1 Noisy Data in der Gegenüberstellung zu Anomalien

Zusammenfassend gilt daher: Anomalieerkennung bedeutet, zuverlässig zwischen relevantem und irrelevantem Verhalten unterscheiden zu können. Dies gelingt am besten, wenn man zunächst klar definiert, was typischerweise normal ist, um anschließend alle deutlichen Abweichungen davon automatisiert und gezielt zu erfassen. Die Definition des Normalverhaltens erfolgt dabei häufig durch statistische Analysen oder durch maschi-

nelles Lernen, wobei etwa Schwellen- oder Grenzwerte festgelegt werden, die bestimmte Kennzahlen nicht überschreiten dürfen. Alternativ können Modelle trainiert werden, die Muster im Normalverhalten erkennen und Abweichungen davon als potenzielle Anomalien markieren. Nur wenn diese Klarheit besteht, erzeugen KI-basierte Systeme wirklichen Mehrwert und unterstützen Entscheidungen zuverlässig – ganz gleich, ob im Finanzbereich, der Produktion, der Logistik oder in anderen Geschäftsbereichen.

4.2.3 Den passenden Algorithmus auswählen

Vielfalt statt Standardlösung

Wenn Sie einen geeigneten Algorithmus zur Anomalieerkennung finden möchten, können Sie aus einer großen Vielfalt potenzieller Methoden wählen. Von einfachen statistischen Verfahren über Machine-Learning-basierte Ansätze bis hin zu komplexen Deep-Learning-Modellen gibt es zahlreiche Möglichkeiten, Auffälligkeiten in Daten automatisiert zu identifizieren. Dabei sollten Sie sich vor Augen führen: Den einen perfekten Algorithmus gibt es nicht. Die Wahl hängt immer von der jeweiligen Problemstellung, dem Umfang und der Qualität der Daten sowie den konkreten Anforderungen des Unternehmens ab.

Klassische statistische Verfahren

Zu den traditionellen statistischen Methoden zählen zum Beispiel Verfahren wie Mittelwertvergleiche, Z-Score-Analysen, Standardabweichungen oder Benford-Analysen. Diese Verfahren sind besonders effektiv, wenn die Daten weitgehend stabil sind und klare, statistisch messbare Grenzwerte existieren.

Vor- und Nachteile

Statistische Verfahren sind schnell implementierbar und leicht interpretierbar. Allerdings funktionieren sie nur bei definierten und stationären Datenmustern. Dynamische oder komplexe Datenmuster werden oft nur unzureichend erkannt. Klassische statistische Verfahren eignen sich beispielsweise für die Erkennung von ungewöhnlich hohen Überweisungen, wenn diese regelmäßig festgelegte Grenzwerte (z. B. dreifache Standardabweichung vom Durchschnittsbetrag) überschreiten.

ML-Verfahren als Allrounder

Algorithmen wie Decision Trees, Random Forests, XGBoost oder Isolation Forest haben sich als besonders vielseitige Ansätze erwiesen und dominieren sogar oft in Hackathons oder Data Challenges. Diese Modelle lernen komplexe Muster und Abweichungen aus historischen Daten. Im Folgenden stellen wir Ihnen diese vier Ansätze vor.

Decision Trees und Random Forests

Decision Trees und *Random Forests* nutzen baumartige Entscheidungsregeln, um Daten zu klassifizieren. Ein Decision Tree trifft Entscheidungen anhand einfacher Ja-/Nein-Fragen (»Ist der Betrag größer als 100 €?«, »Erfolgt die Zahlung nachts?«). Random Forests erweitern dieses Prinzip, indem

mehrere Entscheidungsbäume kombiniert und deren Ergebnisse aggregiert werden, um robustere und präzisere Ergebnisse zu erzielen (siehe Abbildung 4.2).

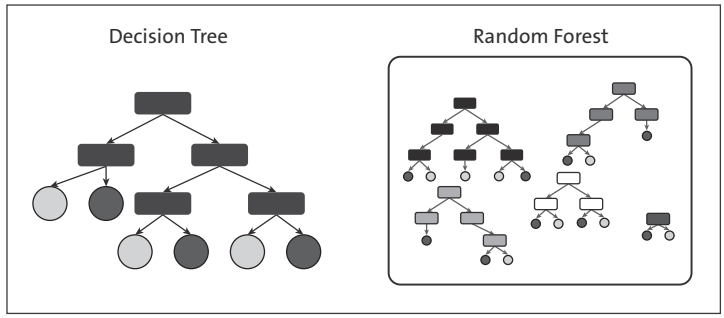

Abbildung 4.2 Decision Tree vs. Random Forest

Einsatzgebiete

Typische Einsatzgebiete sind Kreditwürdigkeitsprüfung, Risikobewertungen, Kundensegmentierungen sowie Entscheidungsunterstützung bei operativen Prozessen wie Kundenservice und Vertragsmanagement. Aufgrund der einfachen Interpretierbarkeit und der schnellen Umsetzung eignen sie sich zudem hervorragend für initiale Machbarkeitsanalysen und Pilotprojekte im Kontext von Predictive Analytics.

Vor- und Nachteile

Die beiden Verfahren treffen transparente und nachvollziehbare Entscheidungen und bieten eine gute Performance insbesondere bei tabellarischen Daten. Allerdings besteht bei ihnen auch die Gefahr des Überanpassens (Overfitting) bei zu kleinen oder zu spezifischen Trainingsdatensätzen.

XGBoost

XGBoost (Extreme Gradient Boosting) ist ein leistungsstarker Machine-Learning-Algorithmus, der auf dem Prinzip des Gradient-Boosting basiert. Dabei werden mehrere schwächere Entscheidungsbäume sequenziell so trainiert, dass jeder neue Baum gezielt die Fehler der vorherigen korrigiert. Dadurch entsteht ein äußerst präzises und leistungsfähiges Modell. XGBoost ist optimal für komplexere und datenintensivere Szenarien, bei denen es auf höchste Genauigkeit und Vorhersageleistung ankommt.

Einsatzbereiche

Typische Einsatzbereiche sind beispielsweise Betrugserkennung in Finanzdaten, Vorhersagen von Kundenverhalten (Churn-Prediction), dynamische Preisanpassungen oder komplexe Risikovorhersagen. XGBoost zeigt seine Stärke besonders dann, wenn sehr umfangreiche historische Datensätze mit vielen Merkmalen vorliegen. Allerdings erfordert der Einsatz von XGBoost ein gewisses Maß an Erfahrung beim Tuning und bei der Optimierung der Modelle, um bestmögliche Ergebnisse zu erzielen.

Vor- und Nachteile

Zu den Vorteilen von XGBoost gehören eine hohe Genauigkeit und Anpassbarkeit und eine besonders hohe Effizienz bei großen Datenmengen. Aller-

dings ist XGBoost auch sehr komplex und bedarf eines höheren Aufwands im Finetuning, also dem Finden der optimalen Parametereinstellungen.

Isolation Forest

Der *Isolation Forest* wurde speziell für die Anomalieerkennung entwickelt und basiert auf dem Konzept, dass Anomalien einfacher und schneller von anderen Datenpunkten isoliert werden können, indem man sie durch zufällige Teilungen der Datenmenge separiert (siehe Abbildung 4.3). Datenpunkte, die sich schnell isolieren lassen, gelten als potenzielle Anomalien. Damit sind sie ideal für Szenarien geeignet, bei denen es primär um die Erkennung ungewöhnlicher Ereignisse geht, ohne dass zuvor definierte auffällige Muster existieren.

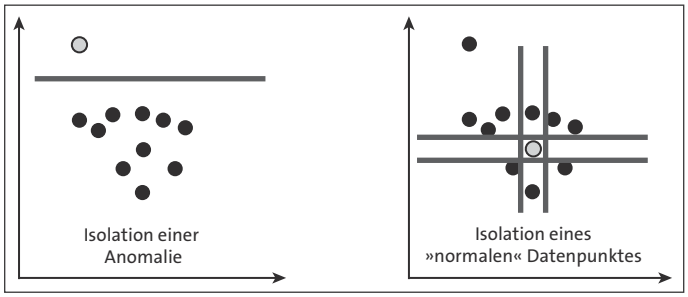

Abbildung 4.3 Darstellung der Funktionsweise eines Isolation Forests

Einsatzbereiche

Konkrete Anwendungsbeispiele sind die Erkennung betrügerischer Finanztransaktionen, das automatische Aufspüren fehlerhafter Buchungen in SAP-Systemen, die Überwachung von Produktionsprozessen auf ungewöhnliche Maschinendaten oder die Analyse von IT-Systemen zur frühzeitigen Erkennung von Cyberangriffen.

Vor- und Nachteile

Der Isolation Forest wurde speziell für die Anomalieerkennung entwickelt und benötigt keine zuvor gelabelten Daten (unsupervised Verfahren). Er verliert bei extrem hochdimensionalen oder stark verrauschten Daten allerdings an Präzision und ist anfällig für Noise (zufällige Schwankungen).

Deep-Learning-Verfahren für komplexere Zusammenhänge

Deep-Learning-Ansätze, insbesondere Autoencoder, rekurrente neuronale Netze (RNN, speziell LSTM) und Transformer-Modelle, eignen sich vor allem für komplexe Datenstrukturen, zeitliche Abfolgen oder sehr große Datenmengen. Diese Algorithmen basieren auf künstlichen neuronalen Netzen mit vielen hintereinander geschalteten Schichten (also tiefen Architekturen), die aus großen Mengen an Trainingsdaten eigenständig Muster und komplexe Zusammenhänge erlernen können.

Autoencoder

Autoencoder sind spezielle neuronale Netze, die darauf ausgelegt sind, Eingabedaten zunächst auf eine stark komprimierte, sogenannte latente Repräsentation zu reduzieren (*Encoding*) und anschließend wieder originalge-

treu zu rekonstruieren (*Decoding*). Autoencoder folgen dem Grundprinzip, typische Muster sehr genau zu rekonstruieren, während Anomalien oder untypische Muster schlecht rekonstruiert werden. Diese Rekonstruktionsfehler werden genutzt, um Anomalien zu erkennen – hohe Fehlerwerte weisen dabei auf eine potenzielle Anomalie hin.

Einsatzgebiete

Typische Einsatzgebiete sind Betrugserkennung im Zahlungsverkehr, bei dem viele verschiedene Transaktionsmerkmale (z. B. Zeit, Ort, Betrag, Händler) gleichzeitig analysiert werden. Auch bei der Überwachung technischer Systeme, beispielsweise von Maschinenparks in der Produktion oder IT-Systemen in Rechenzentren, liefern Autoencoder hervorragende Ergebnisse, indem sie kleinste Abweichungen vom Normalverhalten automatisiert erkennen. Ein weiterer typischer Anwendungsfall ist die Qualitätskontrolle, beispielsweise die automatisierte Inspektion von Produkten mittels Kameras und Bildverarbeitung: Autoencoder erkennen zuverlässig fehlerhafte oder abweichende Produkte allein durch die Analyse visueller Muster.

Vor- und Nachteile

Autoencoder sind sehr effektiv bei komplexen Mustern und hochdimensionalen Daten, sie lernen automatisch, was als typisches Verhalten gilt, ohne dass explizit Anomalien gezeigt werden müssen. Allerdings erfordern sie vergleichsweise große Mengen sauberer Trainingsdaten sowie tiefergehendes technisches Know-how in Bezug auf neuronale Netze und deren Optimierung.

Rekurrente neurale Netze

Rekurrente neuronale Netze (RNNs), insbesondere *Long-Short-Term-Memory-Netze* (LSTM-Netze), sind auf die Verarbeitung sequenzieller oder zeitlicher Daten spezialisiert. Ihre besondere Stärke liegt darin, Zusammenhänge und Muster über Zeiträume hinweg zu erkennen und Informationen im Gedächtnis zu behalten. Das ermöglicht es, untypische Abfolgen oder ungewöhnliche zeitliche Entwicklungen schnell zu identifizieren. Ein LSTM-Netz erkennt beispielsweise, wenn sich ein normalerweise stabiles Zahlungsverhalten einer Kundin oder eines Kunden plötzlich stark verändert, etwa durch unerwartete Transaktionen in ungewöhnlichen zeitlichen Mustern. Sie sind besonders effektiv in Szenarien, in denen zeitliche Abfolgen oder historische Entwicklungen eine entscheidende Rolle spielen.

Einsatzgebiete

Typische Anwendungsfälle sind etwa die kontinuierliche Analyse von Finanztransaktionen, um plötzliche Änderungen in Zahlungsverhalten frühzeitig zu erkennen. Auch die vorausschauende Wartung (Predictive Maintenance) komplexer Maschinen und Anlagen basiert häufig auf LSTM-Netzen, da diese frühzeitig auf ungewöhnliche Abweichungen in Sensordaten reagieren können. In der Logistik eignen sich LSTM-Modelle sehr gut zur Prognose von Nachfrage- oder Lieferkettenstörungen, indem sie kontinu-

ierlich und automatisiert ungewöhnliche Muster in der Lieferhistorie oder im Bestellverhalten erkennen und rechtzeitig Warnungen generieren.

Vor- und Nachteile

LSTM-Netze sind hervorragend geeignet, um Muster in zeitlichen Abfolgen zu erkennen und besonders effektiv bei der frühzeitigen Identifikation von Veränderungen im Verhalten über längere Zeiträume hinweg. Sie haben allerdings auch einen hohen Ressourcenbedarf hinsichtlich Rechenleistung und qualitativ hochwertiger Daten. Sie sind außerdem anspruchsvoll in der Implementierung, Wartung und Optimierung.

Transformer-Modelle

Transformer-Modelle haben in den letzten Jahren insbesondere im Bereich Natural Language Processing (NLP) stark an Bedeutung gewonnen, werden aber zunehmend auch zur Anomalieerkennung eingesetzt. Im Gegensatz zu rekurrenten Netzen verwenden Transformer sogenannte *Attention-Mechanismen*, um Zusammenhänge in Datenfolgen zu erkennen. Dabei berechnen sie, welche Datenelemente in einer Sequenz besonders relevant sind, und richten ihre Aufmerksamkeit gezielt darauf. Gerade bei komplexen zeitlichen Mustern oder großen Datenmengen zeigen Transformer oft eine überlegene Leistung, weil sie Langzeitabhängigkeiten noch präziser und effizienter erkennen können als klassische RNN- oder LSTM-Ansätze.

Einsatzgebiete

Ursprünglich im Bereich des NLP populär geworden, erweisen sie sich zunehmend auch in Bereichen wie Finanzmarktanalysen, Anomalieerkennung in IT-Netzwerken (Cybersecurity) und komplexen Vorhersage-Szenarien wie Absatz- oder Bedarfsplanung als äußerst effektiv. In der Finanzindustrie etwa ermöglichen Transformer die genaue Erkennung ungewöhnlicher Verhaltensmuster, selbst wenn diese über lange Zeiträume verteilt auftreten. Ebenso profitieren Unternehmen mit komplexen Lieferketten von Transformer-Modellen, da sie subtile Veränderungen im globalen Liefernetzwerk präzise erkennen und frühzeitig auf Störungen hinweisen können.

Vor- und Nachteile

Transformer-Modelle bieten eine exzellente Erkennung komplexer Muster und Langzeitabhängigkeiten, eine effiziente Verarbeitung paralleler Daten sowie sehr gute Skalierbarkeit bei großen Datensätzen. Sie erfordern allerdings auch umfangreiche Trainingsdaten und eine hohe Rechenleistung. Die Modelle sind außerdem komplexer zu implementieren und erfordern tiefgehendes technisches Know-how.

LSTM- oder Transformer-Modell?

Die Wahl zwischen LSTM- und Transformer-Modellen hängt stark vom konkreten Anwendungsfall und den verfügbaren Ressourcen ab. Während LSTMs bewährte und robuste Ansätze bieten, glänzen Transformer insbesondere dann, wenn sehr komplexe Zusammenhänge oder besonders lange Zeiträume betrachtet werden sollen.

Den passenden Algorithmus auswählen

Bei der Wahl des richtigen Algorithmus sind stets der konkrete Business-Kontext sowie die verfügbare Datenbasis entscheidend. Ein eher statischer Geschäftskontext mit klar definierten Regeln profitiert von einfachen statistischen Methoden oder klassischen Machine-Learning-Modellen. Komplexe, dynamische Umfelder oder sehr große, unstrukturierte Datenmengen profitieren hingegen stark von Deep-Learning-Verfahren. Häufig zeigt sich in der Praxis, dass eine Kombination mehrerer Ansätze die besten Ergebnisse liefert.

Passgenau statt möglichst komplex

Zusammenfassend gilt deshalb: Nicht der komplexeste Algorithmus gewinnt automatisch, sondern derjenige, der am besten auf das konkrete Geschäftsumfeld, die Qualität und Struktur der Daten und die spezifischen Ziele der Anomalieerkennung abgestimmt ist. Die Auswahl ist daher immer eine bewusste Abwägung der jeweiligen Vor- und Nachteile unter Berücksichtigung der individuellen Rahmenbedingungen.

Datengrundlage der SecureBank AG

Ein gutes Beispiel dafür ist die SecureBank AG: Sie möchte historische Transaktionsdaten mithilfe eines Isolation Forests analysieren, um verdächtige Transaktionen automatisch aufzudecken. Die Entscheidung für dieses vergleichsweise schlanke Verfahren wurde bewusst getroffen – gerade weil die zugrunde liegenden Daten sehr strukturiert vorliegen. Die SecureBank verfügt über eine umfangreiche Historie an Transaktionsdaten, die typischerweise in einem SAP-System gespeichert und verwaltet werden. Diese Daten enthalten Informationen wie den Zeitpunkt der Transaktion (Wochentag, Uhrzeit), den Betrag, die Händlerkategorie (zum Beispiel Lebensmittelgeschäft, Elektronikhändler oaxder Restaurant) sowie die Information, ob es sich tatsächlich um eine betrügerische Transaktion handelte oder nicht (siehe Tabelle 4.1).

transaction_id	amount (€)	weekday	hour	category	is_fraud
521	60.87	Fri	14	Electronics	0
737	69.65	Sat	9	Groceries	0
740	13.18	Mon	11	Electronics	0
411	2731.51	Sun	11	Groceries	1
636	66.46	Sun	14	Electronics	0

Tabelle 4.1 Historischen Transaktionsdaten inklusive Klassifikation (Anomalie/keine Anomalie)

Angesichts dieser klar strukturierten und wenig hochdimensionalen Daten war der Einsatz klassischer Machine-Learning-Verfahren wie Isolation For-

est nicht nur ausreichend, sondern auch effizient. Der Vorteil: Solche Modelle liefern schnelle und gut interpretierbare Ergebnisse bei gleichzeitig geringem Ressourcenaufwand. Für die SecureBank bedeutete dies einen pragmatischen und risikoarmen Einstieg in die Anomalieerkennung – ohne direkt mit »Kanonen auf Spatzen zu schießen«. Dies ist ein Vorgehen, das sich auch für viele andere Unternehmen empfiehlt, bevor in aufwendigere Modellarchitekturen investiert wird.

4.3 Technische Umsetzung

Nachdem Sie nun die theoretischen Grundlagen und Einsatzmöglichkeiten verschiedener Algorithmen zur Anomalieerkennung kennengelernt haben, geht es nun an die praktische Umsetzung.

Inhalt dieses Abschnitts

Wie kann die SecureBank AG nun mithilfe eines Isolation Forests automatisiert Anomalien in ihren Daten erkennen? In den folgenden Abschnitten zeigen wir Schritt für Schritt, wie Sie ein solches Modell vorbereiten, trainieren und effektiv einsetzen können, um betrügerische Transaktionen zu identifizieren, bevor ein Schaden entsteht. Angefangen bei der Datenvorbereitung und Modellentwicklung in Python auf Ihrem lokalen Rechner bis hin zur skalierbaren und cloudbasierten Bereitstellung Ihres Modells als API. Anhand konkreter Codebeispiele, Dockerfiles und Konfigurationsdateien lernen Sie, wie Sie Ihre KI-Lösung stabil und effizient in Ihre SAP-Infrastruktur integrieren und produktiv einsetzen können.

Zunächst wird in Abschnitt 4.3.1 gezeigt, wie ein Isolation Forest in Python implementiert und auf die strukturierten Transaktionsdaten der SecureBank AG angewendet werden kann. Darauf aufbauend wird in Abschnitt 4.3.2 erläutert, wie sich das trainierte Modell in eine REST-API überführen lässt, um es programmgesteuert abfragen zu können. Im anschließenden Abschnitt 4.3.3 bereiten wir eine Docker-Umgebung vor, die sowohl das Training als auch das Deployment der API kapselt und damit eine saubere und wiederholbare Ausführung gewährleistet. In Abschnitt 4.3.4 zeigen wir, wie sich diese Umgebung in automatisierte Workflows überführen lässt – mithilfe von ArgoFlows in SAP AI Core. Den Abschluss bildet Abschnitt 4.3.5, in dem Sie lernen, wie Sie den so erstellten KI-Service mit KServe produktiv in SAP AI Core bereitstellen und damit nahtlos in bestehende Geschäftsprozesse integrieren können.

So entsteht ein vollständiger End-to-End-Prozess – von der ersten Idee bis zur skalierbaren und wartbaren KI-Lösung im SAP-Umfeld.

Lokale Entwicklung vs. Cloud-Entwicklung bei KI-Modellen

Bei der Entwicklung von KI-Modellen stellt sich oft die Frage: Sollte ich zunächst lokal arbeiten oder direkt auf eine cloudbasierte Umgebung wie die SAP Business Technology Platform (BTP) setzen?

Lokale Entwicklung bietet sich vor allem in frühen Projektphasen an, wenn Sie schnell erste Experimente durchführen, verschiedene Algorithmen ausprobieren und unmittelbare Ergebnisse sehen möchten. Lokales Arbeiten ist ideal für kleinere Datensätze, Prototypenentwicklung und schnelles, iteratives Arbeiten – und es benötigt anfangs keine aufwendige Einrichtung oder Cloud-Infrastruktur.

Cloud-Entwicklung empfiehlt sich, sobald die Datensätze groß werden, komplexe Modelle trainiert werden müssen oder eine Integration in produktive SAP-Umgebungen geplant ist. Die Cloud bietet Ihnen skalierbare Ressourcen, professionelle Infrastruktur und nahtlose Integration in bestehende Unternehmensprozesse. Außerdem erleichtert sie kollaborative Arbeit und sorgt für bessere Nachvollziehbarkeit und Wartbarkeit Ihrer KI-Lösungen.

Faustregel: Starten Sie lokal, um Ideen schnell zu validieren, und wechseln Sie zur Cloud, sobald es darum geht, Ihr KI-Modell produktiv, skalierbar und dauerhaft in Ihre Geschäftsprozesse einzubinden.

Dabei startet die SecureBank AG zunächst lokal: Das bedeutet, wir entwickeln, trainieren und evaluieren den Isolation Forest zunächst in einer lokalen Umgebung, beispielsweise auf einem Notebook oder PC. Dieser lokale Start ermöglicht es, schnell erste Ergebnisse zu erzielen, Parameter einfach anzupassen und unmittelbares Feedback zur Modellqualität zu erhalten.

4.3.1 Isolation Forest implementieren

Bibliotheken importieren

Wir starten mit der Implementierung eines Algorithmus, in diesem Fall Isolation Forests, in Python. Ziel ist die Erstellung und das Training eines Modells, das in der Lage ist, Anomalien in Transaktionsdaten zu erkennen. Wie im Python-Universum generell üblich, können verschiedene Pakete genutzt werden, um bestimmte Funktionalitäten abzubilden (siehe Listing 4.1).

```
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import LabelEncoder
import pickle
```

Listing 4.1 Bibliotheken importieren

Python-Bibliotheken

Importieren Sie diese notwendige Python-Bibliotheken:

- `pandas` für die Datenverarbeitung und Verwaltung in tabellarischer Form
- `IsolationForest` (aus `scikit-learn`) als Algorithmus zur Anomalieerkennung
- `LabelEncoder` (ebenfalls aus `scikit-learn`) zur Umwandlung kategorischer Daten in numerische Werte
- `pickle`, um das trainierte Modell und Encoder zu speichern und später wiederverwenden zu können

CSV-Datensatz laden

Laden Sie den CSV-Datensatz mit synthetischen Finanztransaktionen und speichern Sie ihn in einem DataFrame `df`:

```
df = pd.read_csv("/app/src/synthetic_financial_transactions.csv")
```

Dies bildet die Grundlage für die nachfolgende Analyse und Modellentwicklung.

Kategorische Daten umwandeln

Der Isolation-Forest-Algorithmus benötigt numerische Eingabedaten. Übersetzen Sie deshalb kategorische Merkmale wie `weekday` (Wochentag) und `merchant_category` (Händlerkategorie) mittels `LabelEncoder` in numerische Werte (siehe Listing 4.2).

```
le_weekday = LabelEncoder()
le_category = LabelEncoder()

df['weekday_enc'] = le_weekday.fit_transform(df['weekday'])
df['merchant_category_enc'] = le_category.fit_transform(df['merchant_
category'])
```

Listing 4.2 Vorverarbeitung – kategorische Daten umwandeln

Relevante Features auswählen

Speichen Sie diese neuen numerischen Features anschließend als zusätzliche Spalten im `DataFrame` (siehe Listing 4.3).

```
features = ['amount', 'hour', 'weekday_enc', 'merchant_category_enc']
X = df[features]
```

Listing 4.3 Relevante Features auswählen

Isolation Forest initialisieren und trainieren

Wählen Sie nun aus dem vollständigen Datensatz gezielt diejenigen Spalten aus, die das KI-Modell verwenden soll, um Anomalien zu erkennen (siehe Listing 4.4). Das Ergebnis ist eine Feature-Matrix `X`, die für das Training verwendet wird.

```
iso_forest = IsolationForest(contamination=0.02, random_state=42)
iso_forest.fit(X)
```

Listing 4.4 Isolation Forest initialisieren und trainieren

Der Isolation Forest wird in diesem Fall mit zwei Parametern initialisiert:

- `contamination=0.02`: Gibt an, welcher Anteil der Daten als anomal erwartet wird (in diese Fall 2 %).
- `Random_state=42`: Sorgt für die Reproduzierbarkeit der Ergebnisse durch das Setzen eines Seeds.

Modell trainieren

Anschließend trainieren Sie das Modell mit den ausgewählten Features, wodurch es lernt, was typische Muster in den Transaktionen sind.

Modell speichern

Um das trainierte Modell nicht bei jedem Einsatz erneut trainieren zu müssen, wird es mittels `pickle` dauerhaft gespeichert (also persistiert). Neben dem Modell werden auch die verwendeten `LabelEncoder` gespeichert, um später neue Daten entsprechend zu verarbeiten (siehe Listing 4.5).

```
with open("/app/src/isolation_forest_model.pkl", "wb") as f:
    pickle.dump(iso_forest, f)

with open("/app/src/label_encoders.pkl", "wb") as f:
    pickle.dump({'le_weekday': le_weekday, 'le_category':
    le_category}, f)
```

Listing 4.5 Modell speichern (Persistenz)

Vorhersagen generieren

In Listing 4.6 sehen Sie, wie das gespeicherte Modell wieder geladen und auf neue oder bestehende Daten angewendet werden kann. Das Modell generiert hierbei eine Vorhersage (`anomaly_score`):

- `-1` bedeutet, dass die Transaktion als Anomalie erkannt wurde.
- `1` bedeutet, dass die Transaktion normal ist.

Zusätzlich wird eine binäre Spalte `anomaly_detected` eingeführt, die eine schnelle Übersicht darüber gibt, ob eine Anomalie erkannt wurde (1) oder nicht (0).

```
with open("/app/src/isolation_forest_model.pkl", "rb") as f:
    loaded_model = pickle.load(f)

with open("/app/src/label_encoders.pkl", "rb") as f:
    encoders = pickle.load(f)

le_weekday = encoders['le_weekday']
```

```
le_category = encoders['le_category']

df['anomaly_score'] = loaded_model.predict(X)
df['anomaly_detected'] = (df['anomaly_score'] == -1).astype(int)
print(df.head())
print("Training abgeschlossen!")
```

Listing 4.6 Modell laden und anwenden (optional)

Das gesamte Skript in der Übersicht

In Listing 4.7 sehen Sie das gesamte Skript mit allen besprochenen Elementen. Sie finden es ebenfalls im Download-Material dieses Buchs unter *www.sap-press.de/6149*.

```
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import LabelEncoder
import pickle

# 1. Daten laden
df = pd.read_csv("/app/src/synthetic_financial_transactions.csv")

# 2. Vorverarbeitung: Kategorische Daten in Zahlen umwandeln
le_weekday = LabelEncoder()
le_category = LabelEncoder()
df['weekday_enc'] = le_weekday.fit_transform(df['weekday'])
df['merchant_category_enc'] = le_category.fit_transform(df['merchant_
category'])

# 3. Features für das Modell auswählen
features = ['amount', 'hour', 'weekday_enc', 'merchant_category_enc']
X = df[features]

# 4. Isolation Forest initialisieren und trainieren
iso_forest = IsolationForest(contamination=0.02, random_state=42)
iso_forest.fit(X)

# 5. Modell mit pickle speichern
with open("/app/src/isolation_forest_model.pkl", "wb") as f:
    pickle.dump(iso_forest, f)

# 6. LabelEncoder ebenfalls speichern
with open("/app/src/label_encoders.pkl", "wb") as f:
    pickle.dump({'le_weekday': le_weekday, 'le_category':
    le_category}, f)
```

```
# 7. Optional: Modell laden und anwenden
with open("/app/src/isolation_forest_model.pkl", "rb") as f:
    loaded_model = pickle.load(f)
with open("/app/src/label_encoders.pkl", "rb") as f:
    encoders = pickle.load(f)

le_weekday = encoders['le_weekday']
le_category = encoders['le_category']

df['anomaly_score'] = loaded_model.predict(X)
df['anomaly_detected'] = (df['anomaly_score'] == -1).astype(int)
print(df.head())
print("Training abgeschlossen!")
```

Listing 4.7 Gesamtes Skript zur Erstellung und zum Training des Isolation Forests (»predictor.py«)

Optinal: S3-Bucket verwenden

In diesem einfachen Beispiel wird das Modell zunächst lokal gespeichert. Möglich wäre es auch, das Modell nach dem abgeschlossenen Training in einen Object Storage (S3-Bucket) in die SAP BTP zu laden, um das Modell dort in weiteren Services zu nutzen. Gleichermaßen könnten auch die Daten nicht lokal, sondern aus einem S3-Bucket geladen werden.

Dieser Programmcode bietet eine vollständige Pipeline für die Erkennung von Anomalien in Finanztransaktionen. Er umfasst das Laden und Vorbereiten der Daten, das Training eines Isolation-Forest-Modells, dessen Speicherung sowie die anschließende Nutzung des gespeicherten Modells zur Anomalieerkennung in neuen Daten. Die hier gezeigte Methode ist exemplarisch für den Einsatz von Machine Learning in der Praxis und lässt sich flexibel auf ähnliche Szenarien übertragen.

4.3.2 API entwickeln

Um das trainierte Modell nun wirklich nutzen zu können, wird ein Service oder eine API benötigt, um neue Daten hinsichtlich ihrer Anomalität zu prüfen. Auch dieser Service wird zunächst lokal implementiert, um ihn dann im weiteren Verlauf in der Cloud zu deployen.

Benötigte Python-Bibliotheken laden

Laden Sie zunächst die benötigten Python-Bibliotheken (siehe Listing 4.8):

- `pickle` ermöglicht das Laden des gespeicherten Machine-Learning-Modells.
- `FastAPI` ist ein Framework zur schnellen Erstellung von REST-APIs.

- `pydantic` stellt sicher, dass die übermittelten Daten eine definierte Struktur und Datentypen einhalten.
- `pandas` erleichtert das Handling der Daten in Form von Tabellen (DataFrames).

```
import pickle
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List
import pandas as pd
```

Listing 4.8 Bibliotheken und Module importieren

Isolation-Forest-Modell und LabelEncoder laden

Laden Sie nun das zuvor gespeicherte Isolation-Forest-Modell sowie die `LabelEncoder`, die für die Verarbeitung neuer Transaktionsdaten benötigt werden (siehe Listing 4.9). Dies ermöglicht eine konsistente Verarbeitung von neuen Eingabedaten und gewährleistet, dass neue Transaktionen exakt so verarbeitet werden wie beim ursprünglichen Training.

```
with open("/app/src/isolation_forest_model.pkl", "rb") as f:
    model = pickle.load(f)

with open("/app/src/label_encoders.pkl", "rb") as f:
    encoders = pickle.load(f)

le_weekday = encoders['le_weekday']
le_category = encoders['le_category']
```

Listing 4.9 Isolation-Forest-Modell und LabelEncoder laden

API und Datenstruktur definieren

Initialisieren Sie die API (`FastAPI`). Anschließend definieren Sie mithilfe von `pydantic` zwei Datenmodelle (siehe Listing 4.10):

- `Transaction` beschreibt die Struktur einer einzelnen Finanztransaktion.
- `Transactions` ermöglicht die Verarbeitung einer Liste mehrerer Transaktionen gleichzeitig.

Diese Modelle sorgen dafür, dass alle Daten, die an die API geschickt werden, eine festgelegte und validierte Struktur haben.

```
app = FastAPI()

class Transaction(BaseModel):
    amount: float
    hour: int
```

```
    weekday: str
    merchant_category: str

class Transactions(BaseModel):
    data: List[Transaction]
```

Listing 4.10 API und Datenstruktur definieren

API-Endpunkt definieren

Bei der Definition der API-Endpunkte ist besonders wichtig, dass diese stets über eine Versionsnummer `"/v<Nummer>/"` verfügen müssen (siehe Listing 4.11). Andernfalls sind diese im späteren Deployment nicht nutzbar. Der *Healthcheck-Endpunkt* dient dazu, schnell zu überprüfen, ob die API erreichbar ist und ordnungsgemäß läuft. Dies ist insbesondere in Produktionsumgebungen von großer Bedeutung.

```
@app.get("/v2/health")
def health():
    return {"status": "ok"}
```

Listing 4.11 Healthcheck-Endpunkt der API

Prediction-Endpunkt implementieren

Der *Prediction-Endpunkt* ist ebenfalls von zentraler Bedeutung für die API: Er ermöglicht die eigentliche Anomalieerkennung (siehe Listing 4.12):

- Wandeln Sie zunächst die empfangenen Transaktionen in ein `pandas`-DataFrame um, um eine effiziente Verarbeitung zu ermöglichen.
- Wandeln Sie anschließend die kategorischen Merkmale (`weekday` und `merchant_category`) mithilfe der zuvor geladenen LabelEncoder um. Dadurch wird gewährleistet, dass die Daten exakt auf dieselbe Weise verarbeitet werden wie beim Modelltraining.
- Das Isolation-Forest-Modell generiert anschließend eine Vorhersage (`predict`) für jede Transaktion mit den folgenden Werten:
  - `-1` (Anomalie erkannt)
  - `1` (normale Transaktion)
- Diese Ergebnisse werden dann in eine eindeutige, einfach interpretierbare Binärform (`anomaly_detected`: 1 oder 0) umgewandelt.
- Zuletzt werden sowohl die ursprünglichen Eingabedaten als auch das Ergebnis der Anomalieerkennung strukturiert zurückgegeben.

```
@app.post("/v2/predict")
def predict(transactions: Transactions):
    df = pd.DataFrame([t.dict() for t in transactions.data])
```

```
    # Encodieren der Eingabedaten analog zum Training
    df['weekday_enc'] = le_weekday.transform(df['weekday'])
    df['merchant_category_enc'] = le_category.transform(df['merchant_
    category'])

    X = df[['amount', 'hour', 'weekday_enc',
    'merchant_category_enc']]
    preds = model.predict(X)
    anomaly = (preds == -1).astype(int)

    result = []
    for idx, t in enumerate(transactions.data):
        result.append({
            "input": t.dict(),
            "anomaly_detected": int(anomaly[idx])
        })

    return {"predictions": result}
```

Listing 4.12 Prediction-Endpunkt implementieren

Diese API stellt ein produktionsfähiges Interface zur Verfügung, das es erlaubt, Anomalien in Finanztransaktionen automatisiert zu erkennen. Der bereitgestellte Endpoint `/v2/predict` empfängt Transaktionsdaten im JSON-Format, verarbeitet sie genau wie im Modelltraining und liefert anschließend unmittelbar eine eindeutige Aussage über potenzielle Anomalien.

Das gesamte Skript in der Übersicht

In Listing 4.13 sehen Sie das gesamte Skript mit allen besprochenen Elementen. Sie finden es ebenfalls im Download-Material dieses Buchs unter *www.sap-press.de/6149*.

```
import pickle
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List
import pandas as pd

# Modell und LabelEncoder laden
with open("/app/src/isolation_forest_model.pkl", "rb") as f:
    model = pickle.load(f)
with open("/app/src/label_encoders.pkl", "rb") as f:
    encoders = pickle.load(f)
le_weekday = encoders['le_weekday']
```

```
le_category = encoders['le_category']

app = FastAPI()

# Datenmodell für Input
class Transaction(BaseModel):
    amount: float
    hour: int
    weekday: str
    merchant_category: str

class Transactions(BaseModel):
    data: List[Transaction]

@app.get("/v2/health")
def health():
    return {"status": "ok"}

@app.post("/v2/predict")
def predict(transactions: Transactions):
    # Eingabedaten in DataFrame
    df = pd.DataFrame([t.dict() for t in transactions.data])

    # Encodieren wie im Training
    df['weekday_enc'] = le_weekday.transform(df['weekday'])
    df['merchant_category_enc'] = le_category.transform(df['merchant_
    category'])

    X = df[['amount', 'hour', 'weekday_enc',
    'merchant_category_enc']]
    preds = model.predict(X)
    anomaly = (preds == -1).astype(int)

    # Ergebnisse zurückgeben
    result = []
    for idx, t in enumerate(transactions.data):
        result.append({
            "input": t.dict(),
            "anomaly_detected": int(anomaly[idx])
        })

    return {"predictions": result}
```

Listing 4.13 Skript für die API (»main.py«)

Durch diese API-Lösung können Sie das lokal entwickelte KI-Modell effektiv in eine cloudbasierte Infrastruktur wie die SAP BTP integrieren und nahtlos in bestehende Geschäftsprozesse einbinden.

4.3.3 Docker-Umgebung für das Training und Deployment der API vorbereiten

Dockerfile erstellen

Sie können das Training der Anomalieerkennung nun lokal und manuell starten. Dies kann entweder über die `predictor.py`-Datei oder – und dieser Weg ist flexibler –über einen Docker-Container erfolgen. Der Docker-Container kann dann lokal oder auf der SAP BTP genutzt werden (auf Letzteres gehen wir in Abschnitt 4.4 ein). Dazu muss zunächst ein Dockerfile erstellt werden. Als Basis verwenden wir ein offizielles Python-Image (Version 3.13.1). Damit steht direkt eine stabile Python-Laufzeitumgebung zur Verfügung:

```
FROM python:3.13.1
```

Arbeitsverzeichnis innerhalb des Docker-Images erstellen

Erstellen Sie innerhalb des Docker-Images ein Arbeitsverzeichnis (`/app/src`) und legen Sie es als Standardordner für alle weiteren Befehle fest:

```
WORKDIR /app/src
```

Stellen Sie explizit sicher, dass das benötigte Verzeichnis existiert:

```
RUN mkdir -p /app/src/
```

Alle relevanten lokalen Dateien in das Docker-Image kopieren

Kopieren Sie nun alle relevanten lokalen Dateien in das Docker-Image (siehe Listing 4.14):

- `predictor.py` enthält den Programmcode zum Training oder zur Analyse des Isolation-Forest-Modells.
- `synthetic_financial_transactions.csv` ist der Datensatz für die lokale Entwicklung und Tests.
- `requirements.txt` ist die Liste aller benötigten Python-Pakete und Abhängigkeiten.

```
COPY predictor.py ./
COPY synthetic_financial_transactions.csv ./
COPY requirements.txt ./
```

Listing 4.14 Dateien in das Docker-Image kopieren

Notwendige Python-Bibliotheken installieren

Installieren Sie automatisch alle notwendigen Python-Bibliotheken, die in `requirements.txt` angegeben sind, sodass das Modell problemlos ausgeführt werden kann:

```
RUN pip3 install -r requirements.txt
```

Berechtigungen setzen

Setzen Sie die Berechtigungen für den Ordner `/app` so, dass jeder Nutzer im Container Zugriff erhält (zugegeben, ein sehr offener Ansatz für lokale Entwicklungszwecke). Dies ist nützlich, um Dateizugriffe und Entwicklungstests einfacher durchzuführen:

```
RUN chgrp -R 65534 /app && \
    chmod -R 777 /app
```

Inhalt des Dockerfiles

Dieses Dockerfile kann nun genutzt werden, um entsprechend ein Docker-Image aufzubauen und den zugehörigen Container ausführen, um das Training des Modells zu starten. Äquivalent dazu kann auch beim Deployment des fertigen Service (API) vorgegangen werden. Sie finden den Inhalt des Dockerfiles in Listing 4.15.

```
from python:3.13.1

WORKDIR /app/src

# Directory in Ihrem Docker-Image erstellen
RUN mkdir -p /app/src/
#
# File aus Ihrem lokalen System zum Pfad im Docker-Image kopieren
COPY predictor.py ./
COPY synthetic_financial_transactions.csv ./
COPY requirements.txt ./
#
# Abhängigkeiten innerhalb Ihres Docker-Image installieren
RUN pip3 install -r requirements.txt
#
# Berechtigung für den Zugriff auf den Ordner /app aktivieren
RUN chgrp -R 65534 /app && \
    chmod -R 777 /app
```

Listing 4.15 Gesamter Inhalt des Dockerfiles

Docker-Image aufbauen

Bauen Sie das Docker-Image auf (siehe Listing 4.16):

- Basisimage (`FROM python:3.13.1`): Analog zum ersten Dockerfile wird hier ebenfalls ein Python-Standardimage in der Version 3.13.1 genutzt.

- Arbeitsverzeichnis definieren (`WORKDIR /app/src`): Erneut wird das Verzeichnis `/app/src` festgelegt, das als zentrale Stelle für die Anwendung dient.
- Verzeichnis anlegen (`RUN mkdir -p /app/src/`): So wird sichergestellt, dass das Arbeitsverzeichnis existiert.

```
FROM python:3.13.1

WORKDIR /app/src

RUN mkdir -p /app/src/

COPY main.py ./
COPY isolation_forest_model.pkl ./
COPY label_encoders.pkl ./
COPY requirements.txt ./

RUN pip3 install -r requirements.txt

RUN chgrp -R nogroup /app && \
    chmod -R 777 /app
```

Listing 4.16 Docker-Image aufbauen

Dateien in das Docker-Image übertragen

Übertragen Sie die folgende Dateien in das Docker-Image (siehe Listing 4.17):

- `main.py`: Die FastAPI-basierte API zur Echtzeit-Anomalieerkennung.
- `isolation_forest_model.pkl`: Das bereits trainierte Modell.
- `label_encoders.pkl`: Die Encoder zur Verarbeitung neuer Transaktionsdaten.
- `requirements.txt`: Enthält alle notwendigen Abhängigkeiten (z. B. `FastAPI`, `sklearn`, `pandas`), um die API produktiv laufen zu lassen.

```
COPY main.py ./
COPY isolation_forest_model.pkl ./
COPY label_encoders.pkl ./
COPY requirements.txt ./
```

Listing 4.17 Dateien für den produktiven Betrieb kopieren

Installieren Sie alle Python-Bibliotheken, die für die API erforderlich sind:

```
RUN pip3 install -r requirements.txt
```

Offene Zugriffsrechte setzen

Ähnlich wie beim ersten Dockerfile setzen Sie nun offene Zugriffsrechte, allerdings mit der Benutzergruppe `nogroup`, die in vielen Linux-Distributionen als Standardgruppe für nicht zugewiesene Nutzer gilt. Dies erleichtert insbesondere die problemlose Ausführung in cloudbasierten oder containerisierten Umgebungen:

```
RUN chgrp -R nogroup /app && \
    chmod -R 777 /app
```

Inhalt des zweiten Dockerfiles

Den Inhalt des zweiten Dockerfiles sehen Sie in Listing 4.18.

```
from python:3.13.1

WORKDIR /app/src

# Directory in Ihrem Docker-Image erstellen
RUN mkdir -p /app/src/
#
# File aus Ihrem lokalen System zum Pfad im Docker-Image kopieren
COPY main.py ./
COPY isolation_forest_model.pkl ./
COPY label_encoders.pkl ./
COPY requirements.txt ./
#
# Abhängigkeiten innerhalb Ihres Docker-Image installieren
RUN pip3 install -r requirements.txt
#
# Berechtigung für den Zugriff auf den Ordner /app aktivieren
RUN chgrp -R nogroup /app && \
    chmod -R 777 /app
```

Listing 4.18 Gesamter Inhalt des zweiten Dockerfiles

Beide Dockerfiles unterstützen also eine flexible, effiziente und skalierbare Entwicklung sowie den nahtlosen Übergang von lokaler Entwicklung hin zur cloudbasierten Bereitstellung der Lösung für die Anomalieerkennung:

- **Dockerfile 1 (für die lokale Entwicklung)**
  Ideal für schnelles Entwickeln, Testen und Experimentieren mit Daten und Modellvarianten.
- **Dockerfile 2 (für die API/Produktion)**
  Geeignet für den stabilen, produktiven Einsatz einer REST-API zur automatisierten Anomalieerkennung, etwa auf der SAP BTP oder lokal.

Docker-Registry nutzen

Beide Dockerfiles lassen sich entweder lokal ausführen oder – was oftmals der professionellere Ansatz ist – als Image in eine Docker-Registry hochladen. In vielen Unternehmen existiert bereits eine interne Docker-Registry; alternativ kann auch eine öffentliche Registry wie beispielsweise Docker Hub genutzt werden.

Liegt das Dockerfile im aktuellen Verzeichnis, lässt sich das Image mit folgendem Befehl im Terminal erstellen:

```
docker build -t username/imagename:tag .
```

Anmelden

Dabei müssen `username`, `imagename` und `tag` durch eigene Werte ersetzt werden. Nach einer Registrierung auf *hub.docker.com* erfolgt die Anmeldung über:

```
docker login
```

Docker-Image hochladen

Anschließend kann das erstellte Image mit dem folgenden Befehl in die Registry hochgeladen werden:

```
docker push username/imagename:tag
```

Sobald das Image erfolgreich gepusht wurde, steht es auch anderen Ressourcen zur Verfügung.

4.3.4 Automatisierte KI-Workflows mit ArgoFlows in SAP AI Core erstellen

Nachdem die SecureBank AG den Entwicklungsprozess zunächst lokal gestartet hatte, ist nun ein entscheidender Meilenstein erreicht: Das Anomalieerkennungsmodell auf Basis des Isolation Forests wurde erfolgreich entwickelt und in eine lauffähige API überführt. Die Kernkomponenten der Lösung liegen damit vor. An diesem Punkt im Projekt haben Sie also bereits die lokale Entwicklung und Testphase abgeschlossen und verfügen über eine funktionierende technische Grundlage.

Im nächsten Schritt geht es darum, diese Komponenten in den produktiven Betrieb zu überführen. Das bedeutet, Sie stehen nun vor der Aufgabe, Ihr lokal erstelltes Modell und die dazugehörige API in eine skalierbare, cloudbasierte Umgebung zu bringen. Dabei spielt der SAP AI Core als zentrale Runtime-Plattform eine entscheidende Rolle. In diesem Abschnitt zeigen wir Ihnen, wie Sie die zuvor entwickelten Komponenten auf die SAP Business Technology Platform (SAP BTP) übertragen und dort betreiben können – damit Ihre KI-Lösung zuverlässig, skalierbar und wartbar im Unternehmenskontext eingesetzt werden kann.

ArgoFlows

In SAP AI Core spielen sogenannte *ArgoFlows* eine zentrale Rolle, wenn es darum geht, komplexe KI-Workflows zuverlässig, wiederholbar und skalierbar zu automatisieren. ArgoFlows basieren auf der Open-Source-Technologie *Argo Workflows* und ermöglichen die präzise Steuerung sämtlicher Schritte im Lebenszyklus von KI-Modellen – vom Training bis zur produktiven Bereitstellung. Mithilfe von YAML-basierten Workflow-Templates lässt sich definieren, wie beispielsweise ein Isolation-Forest-Modell automatisiert trainiert und verwaltet werden kann. Diese Templates beschreiben detailliert, welche Container ausgeführt werden, wie Daten verarbeitet werden und wie SAP AI Core nahtlos mit Docker-Images interagiert. Dadurch werden manuelle Eingriffe auf ein Minimum reduziert und höchste Effizienz bei der Entwicklung und beim Betrieb von KI-Lösungen sichergestellt. SAP stellt dabei Workflow-Templates bereit, die nach den eigenen Anforderungen angepasst werden können.

Training des Isolation Forests

Hier ist ein Beispiel für das Training des Isolation Forests:

```
apiVersion: argoproj.io/v1alpha1
kind: WorkflowTemplate
```

`apiVersion` definiert hier die verwendete Version der Argo-Workflows-API, die SAP AI Core zugrunde liegt. `kind` gibt den Typ des Templates an (`WorkflowTemplate`), das in SAP AI Core verwendet wird, um wiederverwendbare Workflow-Abläufe zu definieren.

Definieren Sie die Metadaten des Workflows (siehe Listing 4.19). Der Parameter `name` definiert einen eindeutigen Namen für den Workflow – in diesem Fall `train-predictor`. Dieser Name muss eindeutig sein, insbesondere wenn mehrere Workflows parallel in SAP AI Core betrieben werden, da er der zentrale Identifikator für die Ausführung und Verwaltung ist.

»annotations«

Unter `annotations` werden zusätzliche beschreibende Informationen hinterlegt:

- `scenarios.ai.sap.com/description` bietet eine kurze Beschreibung des Zwecks des Workflows.
- `scenarios.ai.sap.com/name` benennt das übergeordnete Szenario, dem der Workflow zugeordnet ist.
- `executables.ai.sap.com/description` beschreibt die konkrete Aufgabe des Workflows – in diesem Fall das Training eines Isolation-Forest-Modells.
- `executables.ai.sap.com/name` benennt den Anwendungsfall klar als »Financial Transaction Anomaly«.

»labels«

Die unter `labels` angegebenen Metadaten helfen dabei, Workflows in SAP AI Core systematisch zu organisieren, zu versionieren und effizient zu verwalten. Sie ermöglichen eine strukturierte Zuordnung und eine bessere Nachvollziehbarkeit in komplexeren Umgebungen mit vielen Szenarien.

```
metadata:
  name: train-predictor
  annotations:
    scenarios.ai.sap.com/description: "Tutorial to add custom code to
SAP AI Core"
    scenarios.ai.sap.com/name: "Code (AI Business Cases)"
    executables.ai.sap.com/description: "Trains Isolation Forest"
    executables.ai.sap.com/name: "Financial Transaction Anomaly
(Sklearn Example)"
  labels:
    scenarios.ai.sap.com/id: "train-predictor"
    ai.sap.com/version: "3.0"
```

Listing 4.19 Metadaten (Namen und Annotationen)

»imagePullSecrets«

Legen Sie nun die Workflow-Spezifikationen fest (siehe Listing 4.20). Der Parameter `imagePullSecrets` referenziert ein sogenanntes Secret, das die Zugangsdaten zu einer privaten Docker-Registry enthält. Dieses Secret wird benötigt, damit SAP AI Core das angegebene Docker-Image aus einer geschützten Registry abrufen kann. Ohne diese Angabe könnte das Image nicht geladen werden, wenn es nicht öffentlich zugänglich ist.

»entrypoint«

Der Eintrag `entrypoint` legt fest, welcher Teil des Workflows beim Start ausgeführt werden soll. In diesem Fall ist `mypipeline` der Einstiegspunkt – also die benannte Pipeline, mit der der Workflow beginnt. Diese Pipeline definiert die konkreten Schritte, die im Rahmen der Ausführung nacheinander abgearbeitet werden.

```
spec:
  imagePullSecrets:
    - name: ai-business-cases
  entrypoint: mypipeline
```

Listing 4.20 Workflow-Spezifikation (»spec«)

»mypipeline«

Definieren Sie nun die Template- und Pipeline-Struktur (siehe Listing 4.21). Der Abschnitt `mypipeline` beschreibt die eigentliche Pipeline, die den Ablauf des Workflows definiert. In dieser Pipeline wird festgelegt, dass als Erstes ein Schritt namens `mypredictor` ausgeführt wird. Dieser verweist auf das Tem-

plate `mycodeblock1`, das wiederum die Konfiguration des Containers und dessen auszuführenden Befehl enthält.

»steps« Die Struktur basiert auf einem hierarchischen Aufbau: Innerhalb von `steps` können eine oder mehrere Aktionen nacheinander oder parallel ausgeführt werden. Diese Form der Definition erlaubt es, auch komplexere, mehrstufige Workflows abzubilden, beispielsweise für das sequenzielle Training, Testen und Bereitstellen von KI-Modellen oder für die parallele Ausführung von Vorverarbeitungsschritten.

```
templates:
  - name: mypipeline
    steps:
    - - name: mypredictor
        template: mycodeblock1
```

Listing 4.21 Template- und Pipeline-Struktur

»mycodeblock1« Der Abschnitt `mycodeblock1` beschreibt die konkrete Ausführungseinheit innerhalb des Workflows. Hier wird der Container definiert, in dem das KI-Modelltraining durchgeführt wird. Dieser Container bildet das Herzstück der Pipeline, da er den eigentlichen Programmcode zur Anomalieerkennung enthält.

»image« Mit dem Parameter `image` wird das Docker-Image angegeben, das in SAP AI Core verwendet werden soll. In diesem Fall handelt es sich um das Image `docker.io/<user>/anomaly-predictor:03`, das den vollständigen Python-Code (`predictor.py`), alle notwendigen Abhängigkeiten sowie – je nach Aufbau – auch Trainingsdaten und gegebenenfalls ein initiales Modell beinhaltet.

»command« und »args« Als Nächstes legen Sie die Container-Definition fest und führen das Modell aus (siehe Listing 4.22). Die Einträge `command` und `args` geben an, was im Container nach dem Start ausgeführt werden soll. Mit `/bin/sh -c` wird eine Shell geöffnet, in der dann das Skript `predictor.py` ausgeführt wird. Dieses Python-Skript enthält die Logik zum Training oder zur Anwendung des Isolation-Forest-Modells und stellt sicher, dass der Anomalieerkennungsprozess vollständig automatisiert im Container abläuft.

```
  - name: mycodeblock1
    container:
      image: docker.io/<user>/anomaly-predictor:03
      command: ["/bin/sh", "-c"]
      args:
        - "python /app/src/predictor.py"
```

Listing 4.22 Container definieren und Modell ausführen

Das gesamte Skript in der Übersicht

In Listing 4.23 sehen Sie das gesamte Skript mit allen besprochenen Elementen. Sie finden es ebenfalls im Download-Material dieses Buchs unter *www.sap-press.de/6149*.

```
apiVersion: argoproj.io/v1alpha1
kind: WorkflowTemplate
metadata:
  name: train-predictor # executable id, must be unique across all
your workflows (YAML files), please modify this to any value (e.g.
code-pipeline-12345) if you are not the only user of your SAP AI Core
instance.
  annotations:
    scenarios.ai.sap.com/description: "Tutorial to add custom code to
SAP AI Core"
    scenarios.ai.sap.com/name: "Code (AI Business Cases)"
    executables.ai.sap.com/description: "Trains Isolation Forest"
    executables.ai.sap.com/name: "Financial Transaction Anomaly
(Sklearn Example)"
  labels:
    scenarios.ai.sap.com/id: "train-predictor"
    ai.sap.com/version: "3.0"
spec:
  imagePullSecrets:
    - name: ai-business-cases # your docker registry secret
  entrypoint: mypipeline
  templates:
  - name: mypipeline
    steps:
    - - name: mypredictor
        template: mycodeblock1

  - name: mycodeblock1
    container:
      image: docker.io/<user>/anomaly-predictor:03 # Your docker
image name
      command: ["/bin/sh", "-c"]
      args:
        - "python /app/src/predictor.py"
```

Listing 4.23 Gesamtes Workflow-File (»train-predictor.yaml«)

Diese YAML-Datei bildet eine vollständige Workflow-Vorlage für SAP AI Core ab. Sie automatisiert das Ausführen eines KI-Modells im Kontext von Finanztransaktionsanomalien, das einen Isolation Forest nutzt, und stellt

sicher, dass das Training bzw. die Ausführung sauber und reproduzierbar über Docker-Container abläuft.

Damit kann ein lokal entwickeltes KI-Modell auf einfache Weise professionell und skalierbar in der SAP-AI-Core-Umgebung betrieben werden.

4.3.5 KI-Services mit KServe und SAP AI Core produktiv bereitstellen

KServe

Nachdem in Abschnitt 4.3.4 gezeigt wurde, wie das KI-Modell mithilfe von ArgoFlows erfolgreich trainiert werden kann, rückt die nächste Herausforderung in den Fokus: die produktive Bereitstellung des Modells als skalierbarer Service. SAP AI Core setzt hier auf den KServe-Standard und sogenannte *Serving-Templates*, um den produktiven Einsatz von KI-Modellen in der Cloud zu automatisieren. *KServe* ist ein Open-Source-Framework, das speziell für das Bereitstellen und Skalieren von Machine-Learning-Modellen in Kubernetes-Umgebungen entwickelt wurde.

Die YAML-basierte Definition ermöglicht es, Modell-Container mit wenigen Zeilen Konfiguration als performante und skalierbare REST-API bereitzustellen – etwa auf Basis von FastAPI und Gunicorn. Dank ausgefeilter Autoscaling-Mechanismen und der tiefen Integration in SAP AI Core lassen sich so auch anspruchsvolle Use Cases wie die Anomalieerkennung im Finanzbereich einfach, zuverlässig und zukunftssicher produktiv machen.

»apiVersion«

Der Eintrag `apiVersion` legt die verwendete Version der SAP-AI-Core-API fest: In diesem Fall `v1alpha1`. Diese Versionierung stellt sicher, dass die YAML-Struktur und die enthaltenen Funktionen mit dem jeweiligen Stand der SAP-AI-Core-Plattform kompatibel sind. Gleichzeitig kann so sichergestellt werden, dass Lösungen, die stetig weiterentwickelt werden, auch nachvollziehbar versioniert werden können.

»kind«

Der Parameter `kind` definiert den Typ der Ressource. Mit `ServingTemplate` wird angegeben, dass es sich um eine Vorlage zur Bereitstellung eines Modells handelt. Solche Templates ermöglichen es, ein trainiertes KI-Modell als dauerhaft erreichbaren Service bereitzustellen – beispielsweise in Form einer REST-API – und es so produktiv im Unternehmen zu nutzen:

```
apiVersion: ai.sap.com/v1alpha1
kind: ServingTemplate
```

»name«

Als Nächstes definieren Sie die Metadaten des Modells (siehe Listing 4.24). Der Parameter `name` dient zur eindeutigen Kennzeichnung des Serving-Templates – in diesem Fall unter dem Namen `anomaly-predictor-server`. Dieser Name identifiziert den Bereitstellungsprozess des Modells eindeutig

innerhalb von SAP AI Core und sollte daher konsistent und sprechend gewählt werden.

»annotations«

Die `annotations` enthalten zusätzliche beschreibende Informationen zum Szenario. Sie helfen dabei, den Zweck und Kontext der Bereitstellung schnell zu erfassen – hier geht es um die Bereitstellung eines Anomalieerkennungsmodells auf Basis von FastAPI und Isolation Forest. Diese Informationen werden später in der Benutzeroberfläche von SAP AI Core angezeigt und unterstützen die Dokumentation und Verständlichkeit des Modells.

»labels«

Die `labels` ermöglichen eine strukturierte Kategorisierung, Filterung und Versionierung innerhalb der SAP-AI-Core-Plattform. Sie erleichtern insbesondere in umfangreicheren Projekten mit mehreren Modellen und Versionen die Übersicht und Verwaltung.

```
metadata:
  name: anomaly-predictor-server
  annotations:
    scenarios.ai.sap.com/description: "Anomaly detection with
FastAPI"
    scenarios.ai.sap.com/name: "Anomaly FastAPI"
    executables.ai.sap.com/description: "FastAPI Isolation Forest"
    executables.ai.sap.com/name: "server"
  labels:
    scenarios.ai.sap.com/id: "anomaly-predictor-server"
    ai.sap.com/version: "3.0"
```

Listing 4.24 Metadaten (Name und Annotationen)

»apiVersion«

Spezifizieren Sie nun das Serving-Template (siehe Listing 4.25). Die Angabe der KServe-API-Version (`apiVersion: serving.kserve.io/v1beta1`) zeigt, dass dieses Serving-Template auf dem KServe-Standard basiert. SAP AI Core verwendet KServe als technische Grundlage, um KI-Modelle effizient, skalierbar und standardisiert bereitzustellen. Dadurch lassen sich Modelle als Services betreiben, die bei Bedarf automatisch hoch- oder herunterskaliert werden können – je nachdem, wie viele Anfragen verarbeitet werden müssen. Die Nutzung von KServe ermöglicht damit eine moderne, cloud-native Infrastruktur für produktionsreife KI-Anwendungen.

```
spec:
  template:
    apiVersion: "serving.kserve.io/v1beta1"
```

Listing 4.25 Spezifikation (»spec«) des Serving-Templates

Autoscaling konfigurieren

Anschließend konfigurieren Sie das Autoscaling und die Ressourcenplanung (siehe Listing 4.26). Die Autoscaling-Konfiguration legt fest, wie SAP AI Core auf Basis der Nutzungslast automatisch die Anzahl der laufenden Instanzen (Pods) skaliert:

- `metric: concurrency` bedeutet, dass die Skalierung sich an der Anzahl gleichzeitiger Anfragen orientiert. Wenn mehrere Anfragen parallel verarbeitet werden müssen, kann das System zusätzliche Instanzen starten.
- `target: 1` definiert, dass ein Pod im Durchschnitt eine Anfrage gleichzeitig bedienen soll. Wird dieser Schwellenwert überschritten, wird automatisch ein weiterer Pod gestartet – so wird eine gleichbleibende Antwortzeit sichergestellt.
- `targetBurstCapacity: 0` deaktiviert die sogenannte Burst-Kapazität, d. h., es wird keine zusätzliche Reservekapazität für plötzliche Lastspitzen bereitgehalten. Die Skalierung erfolgt kontrolliert und verzichtet auf spontane Hochskalierung über die definierte Maximalanzahl hinaus.

»resourcePlan«

Die Angabe `resourcePlan: starter` legt fest, welche Ressourcenklasse in SAP AI Core verwendet wird – in diesem Fall das sogenannte Starterpaket, das typischerweise eine begrenzte Menge an CPU und Arbeitsspeicher bereitstellt. Dies ist ideal für Entwicklung, Tests oder kleinere produktive Modelle mit moderatem Ressourcenbedarf.

```
metadata:
  annotations: |
    autoscaling.knative.dev/metric: concurrency
    autoscaling.knative.dev/target: 1
    autoscaling.knative.dev/targetBurstCapacity: 0
  labels: |
    ai.sap.com/resourcePlan: starter
```

Listing 4.26 Autoscaling und Ressourcenplanung

»imagePullSecrets«

Definieren Sie nun den Container, um das Modell bereitzustellen (siehe Listing 4.27). Der Eintrag `imagePullSecrets` ermöglicht den Zugriff auf private oder geschützte Docker-Registries. Über das angegebene Secret kann SAP AI Core das benötigte Docker-Image sicher abrufen, auch wenn es nicht öffentlich zugänglich ist.

»minReplicas« und »maxReplicas«

Mit den Parametern `minReplicas` und `maxReplicas` wird definiert, wie viele Instanzen (Pods) des bereitgestellten KI-Modell-Service mindestens und maximal gleichzeitig laufen dürfen. In diesem Fall bedeutet `minReplicas: 1`, dass der Service dauerhaft verfügbar bleibt, während `maxReplicas: 3` eine au-

tomatische Skalierung bis zu drei parallelen Instanzen erlaubt – abhängig von der aktuellen Auslastung.

»containers«

Im Abschnitt `containers` wird der technische Aufbau des Deployments beschrieben:

- `name` identifiziert den Container eindeutig innerhalb des Serving-Prozesses.
- `image` gibt an, welches Docker-Image verwendet wird. Dieses Image enthält das trainierte Modell, die API-Logik (z. B. mit FastAPI) und alle Abhängigkeiten.
- `ports` definiert, über welchen Port (hier: 9001) der Service erreichbar gemacht wird. Über diesen Port nimmt der Container API-Anfragen entgegen.

»command« und »args«

Die Felder `command` und `args` legen fest, wie der Container beim Start den Webservice ausführt. Hierbei wird mit `/bin/sh -c` zunächst eine Shell geöffnet, in der dann der Befehl

```
gunicorn --chdir /app/src main:app -k uvicorn.workers.UvicornWorker
-b 0.0.0.0:9001
```

»gunicorn« und »uvicorn«

ausgeführt wird. Dabei übernimmt `gunicorn` die Rolle eines robusten und skalierbaren Server-Hosts, der auch im produktiven Umfeld zuverlässig läuft. `uvicorn` fungiert als performanter ASGI-kompatibler Server, der speziell für moderne Python-Webframeworks wie FastAPI entwickelt wurde. Gemeinsam sorgen sie dafür, dass die FastAPI-Anwendung performant, stabil und produktionsfähig betrieben werden kann.

```
    spec: |
      predictor:
        imagePullSecrets:
          - name: ai-business-cases
        minReplicas: 1
        maxReplicas: 3
        containers:
        - name: kserve-container
          image: "docker.io/<user>/anomaly-predictor-server:02"
          ports:
            - containerPort: 9001
              protocol: TCP
          command: ["/bin/sh", "-c"]
          args:
```

```
            - >
              set -e && echo "Starting" && gunicorn --chdir /app/src
main:app -k uvicorn.workers.UvicornWorker -b 0.0.0.0:9001
```

Listing 4.27 Container definieren zur Modellbereitstellung

Das gesamte Skript in der Übersicht

In Listing 4.28 sehen Sie das gesamte Skript mit allen besprochenen Elementen. Sie finden es ebenfalls im Download-Material dieses Buchs unter *www.sap-press.de/6149*.

```
apiVersion: ai.sap.com/v1alpha1
kind: ServingTemplate
metadata:
  name: anomaly-predictor-server
  annotations:
    scenarios.ai.sap.com/description: "Anomaly detection with
    FastAPI"
    scenarios.ai.sap.com/name: "Anomaly FastAPI"
    executables.ai.sap.com/description: "FastAPI Isolation Forest"
    executables.ai.sap.com/name: "server"
  labels:
    scenarios.ai.sap.com/id: "anomaly-predictor-server"
    ai.sap.com/version: "3.0"
spec:
  template:
    apiVersion: "serving.kserve.io/v1beta1"
    metadata:
      annotations: |
        autoscaling.knative.dev/metric: concurrency
        autoscaling.knative.dev/target: 1
        autoscaling.knative.dev/targetBurstCapacity: 0
      labels: |
        ai.sap.com/resourcePlan: starter
    spec: |
      predictor:
        imagePullSecrets:
          - name: ai-business-cases
        minReplicas: 1
        maxReplicas: 3
        containers:
        - name: kserve-container
          image: "docker.io/<user>/anomaly-predictor-server:02"
          ports:
            - containerPort: 9001
```

```
            protocol: TCP
          command: ["/bin/sh", "-c"]
          args:
            - >
              set -e && echo "Starting" && gunicorn --chdir /app/src
main:app -k uvicorn.workers.UvicornWorker -b 0.0.0.0:9001
```

Listing 4.28 Gesamtes Workflow-File für das Ausführen des API-Service (»run_predictor.yaml«)

Dieses Serving-Template erlaubt es, einen KI-basierten Anomalieerkennungsservice, der einen Isolation Forest nutzt, effizient, skalierbar und produktiv in SAP AI Core bereitzustellen. Die Konfiguration nutzt Industriestandards (KServe, Knative Autoscaling, FastAPI, Gunicorn/Uvicorn), um optimale Performance, automatisierte Skalierung und die zuverlässige Bereitstellung der KI-Lösung sicherzustellen.

Durch diese YAML-Datei wird die lokale KI-Lösung nahtlos in der Cloud betrieben und in die bestehenden Unternehmensprozesse integriert.

4.4 Die KI-Lösung auf der SAP BTP bereitstellen

Nach der lokalen Entwicklung des KI-Modells und der technischen Vorbereitung aller relevanten Komponenten – wie Python-Skripte, Docker-Container, YAML-Workflows und Serving-Konfigurationen – folgt nun der entscheidende Schritt: die Überführung in eine produktive Umgebung innerhalb der *SAP Business Technology Platform* (SAP BTP).

Inhalt dieses Abschnitts

In diesem Abschnitt zeigen wir Ihnen, wie Sie Ihre KI-Anwendung auf der SAP BTP operationalisieren. Sie lernen, wie Sie:

- Ihre Code-Artefakte und Container registrieren,
- Workflows in SAP AI Core deployen,
- Servings für Vorhersagen einrichten
- und Ihre Anomalieerkennung als produktionsfähigen Service bereitstellen.

Dabei liegt der Fokus bewusst nicht nur auf der technischen Umsetzung, sondern auch auf den administrativen und operativen Schritten im SAP BTP Cockpit – von der Projektstruktur bis zur Anbindung an das SAP AI Launchpad. Mit dieser Anleitung machen Sie aus einem lokal entwickelten KI-Prototyp eine skalierbare, wartbare und integrierbare Lösung im SAP-Ökosystem.

4.4.1 Administrative Grundlagen in SAP AI Core

Service Key

Bevor ein lokal entwickeltes KI-Modell auf SAP AI Core deployt werden kann, muss ein sogenannter *Service Key* erstellt werden. Dieser Schlüssel enthält alle notwendigen Verbindungsinformationen (z. B. API-URL, Token-Endpunkte, Zugriffstoken), die benötigt werden, um per Skript, CLI oder Workflow-Engine sicher auf den SAP AI Core zuzugreifen.

Service Key erstellen

Im Fall der SecureBank AG gehen wir davon aus, dass der SAP AI Core Service bereits im entsprechenden Space innerhalb der BTP konfiguriert und erfolgreich aktiviert wurde. Die Instanz ist unter dem Namen `aicore` sichtbar und hat den Status **Created**. Ein Klick auf **Create** im Bereich **Service Keys** erzeugt einen neuen Schlüssel, der anschließend exportiert oder direkt für lokale Workflows genutzt werden kann (siehe Abbildung 4.4).

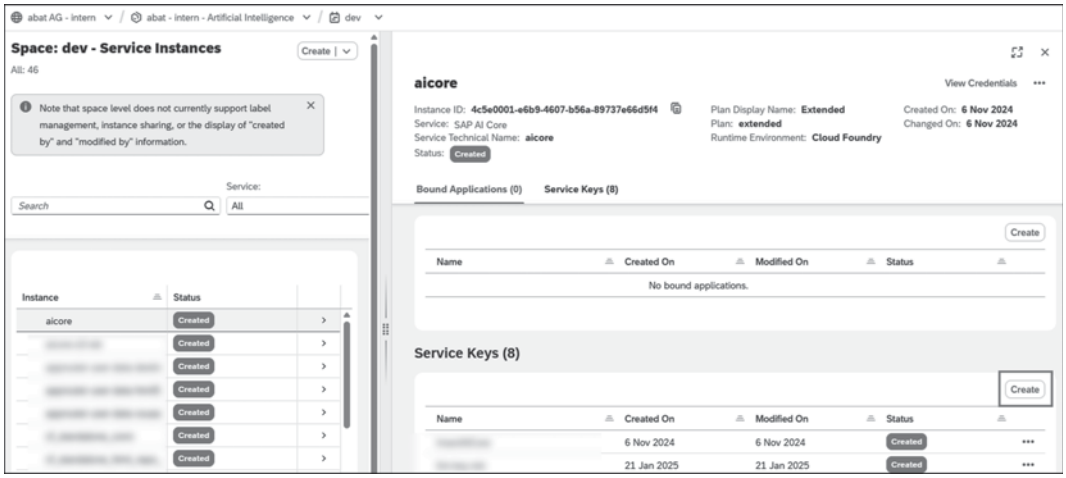

Abbildung 4.4 Service Key erstellen

Service Key im SAP AI Launchpad einbinden

Nachdem Sie den Service Key (Serviceschlüssel) für den SAP AI Core erstellt haben, muss dieser im SAP AI Launchpad eingebunden werden, um den Zugriff auf die API und die bereitgestellten Workflows zu ermöglichen. Hierzu legen Sie eine neue AI-API-Verbindung an.

Benötigte Angaben

In der Maske **AI-API-Verbindung anlegen** benötigen Sie die folgenden Angaben (siehe Abbildung 4.5):

- ein eindeutiger **Verbindungsname** (z. B. »ai-business-cases-conn«)
- der zuvor exportierte **Serviceschlüssel** (z. B. als Textdatei)
- die **Verbindungsart** (in der Regel **Geheimer Schlüssel**)
- der automatisch extrahierte Wert für die **AI-API-URL** – also der Endpunkt zur Kommunikation mit dem AI Core

- der automatisch extrahierte Wert für die **XSUAA-URL**, **Client-ID** und den geheimen **Client-Schlüssel**, die aus dem Serviceschlüssel stammen

AI-API-Verbindung anlegen

Nach dem Klick auf **Anlegen** steht die Verbindung im SAP AI Launchpad zur Verfügung und kann für Trainings, Servings und Deployments verwendet werden.

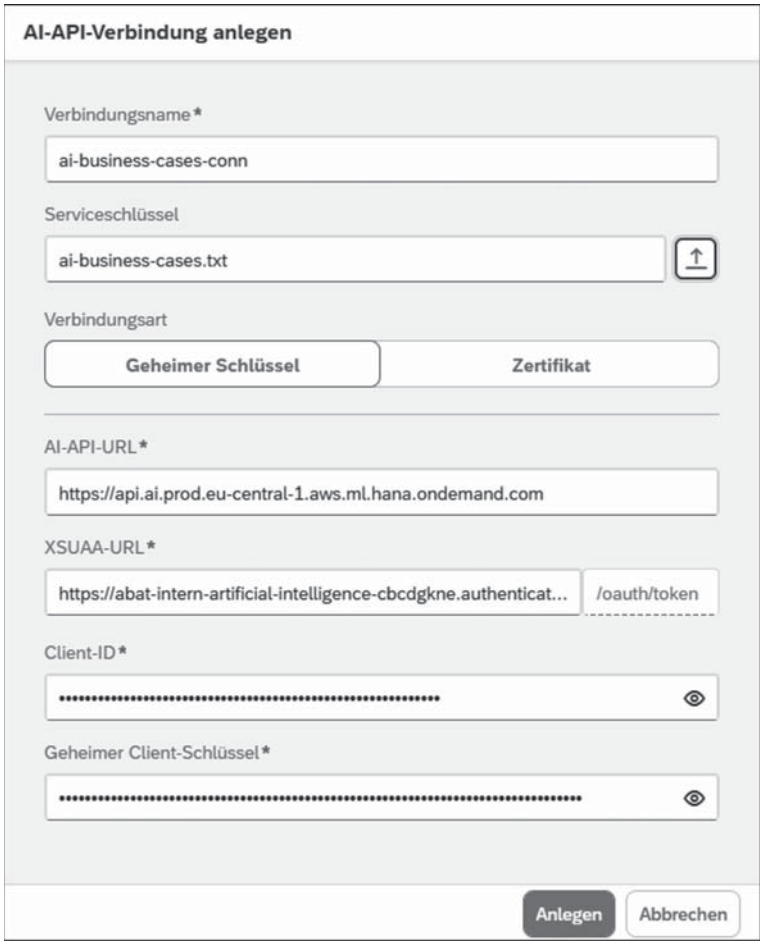

Abbildung 4.5 Neue AI-API-Verbindung erstellen

Im nächsten Schritt verknüpfen Sie Ihr Git-Repository mit dem SAP AI Launchpad. In diesem Repository befinden sich die zuvor bereits entwickelten Workflow-Templates, Serving-Definitionen und Konfigurationsdateien. Durch diese Verbindung werden die Inhalte des Repositorys automatisch in SAP AI Core übernommen und stehen dort zur weiteren Verwendung (z. B. für Trainings oder Deployment-Prozesse) zur Verfügung.

Repository hinzufügen

Fügen Sie dazu im Bereich **SAP-AI-Core-Administration • Git-Repositorys** ein neues Repository hinzu oder bearbeiten Sie ein bestehendes. Die folgenden Angaben sind dabei erforderlich (siehe Abbildung 4.6):

- die vollständige **URL** des Repositorys (z. B. GitHub)
- ein frei wählbarer **Name** zur Identifikation im SAP AI Launchpad
- der persönliche (GitHub-)**Benutzername**
- ein gültiges **Zugriffs-Token**, um SAP AI Core den sicheren Zugriff auf die Repository-Inhalte zu ermöglichen

Sobald der Onboarding-Prozess erfolgreich abgeschlossen ist (erkennbar am Status **COMPLETED**), stehen die Inhalte des Repositorys in SAP AI Core zur Verfügung – etwa zur Ausführung von Trainingsworkflows oder zum Anlegen von Servings.

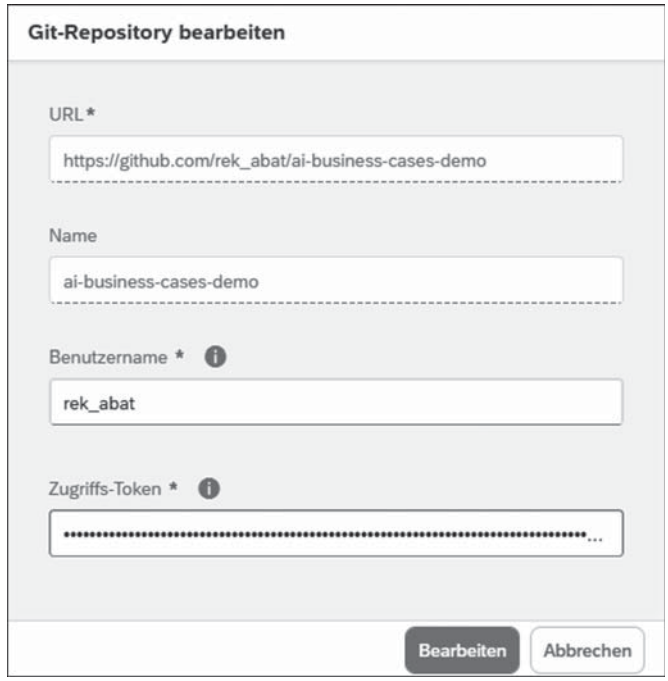

Abbildung 4.6 Git-Repository im SAP AI Launchpad anbinden

Neben dem Zugriff auf das Git-Repository muss SAP AI Core auch auf die Docker-Images zugreifen können, die Ihre trainierten Modelle oder Serv-

ing-Anwendungen enthalten. Dazu ist es notwendig, dass Sie Ihre Docker-Registry mit SAP BTP verbinden – etwa Docker Hub oder eine private Container-Registry.

Geheimen Docker-Registry-Schlüssel hinterlegen

Dafür hinterlegen Sie einen sogenannten *geheimen Docker-Registry-Schlüssel* (Docker Secret) im Bereich **SAP-AI-Core-Administration • Geheime Docker-Registry-Schlüssel** (siehe Abbildung 4.7).

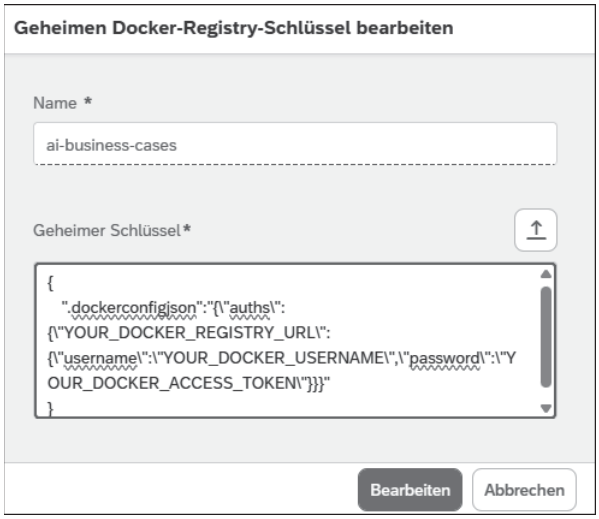

Abbildung 4.7 Geheimen Docker-Registry-Schlüssel in SAP AI Core hinterlegen

Zugangsdaten

Sie benötigen dafür ein JSON-Format mit Zugangsdaten (siehe Listing 4.29).

```
json
{
  ".dockerconfigjson": "{\"auths\": {\"<DOCKER_REGISTRY_URL>\": {\
"username\": \"<DOCKER_USERNAME>\", \"password\": \"<DOCKER_ACCESS_
TOKEN>\"}}}"
}
```

Listing 4.29 Aufbau des JSON für den geheimen Docker-Registry-Schlüssel

Passwort generieren

Das Passwort ist in diesem Fall ein sogenannter *Personal Access Token*, den Sie zuvor direkt in der Docker-Registry (z. B. *hub.docker.com*) generieren müssen. Benutzername und Token ermöglichen SAP AI Core das automatisierte Pullen (also das Herunterladen) der Images bei Workflows und Deployments.

4.4.2 Anwendung registrieren

Sobald das Git-Repository und die Docker-Registry erfolgreich mit der SAP BTP verbunden wurden, synchronisiert SAP AI Core automatisch alle verfügbaren Inhalte aus dem Repository. Im nächsten Schritt müssen Sie nun eine sogenannte *Anwendung* registrieren. Diese verknüpft einen bestimmten Pfad im Repository mit der SAP-AI-Core-Umgebung.

Neue Anwendung erstellen

Dazu navigieren Sie zu **SAP-AI-Core-Administration • Anwendungen** und erstellen eine neue Anwendung. Für die Konfiguration benötigen Sie die folgenden Angaben (siehe Abbildung 4.8):

- einen eindeutigen **Anwendungsname**n
- das gewünschte **Repository** (z. B. »ai-business-cases-demo«)
- den anzugebenden relevanten **Pfad im Repository** (z. B. »run«)
- eine festzulegende **Revision** (z. B. »HEAD« für den neuesten Stand im Haupt-Branch)

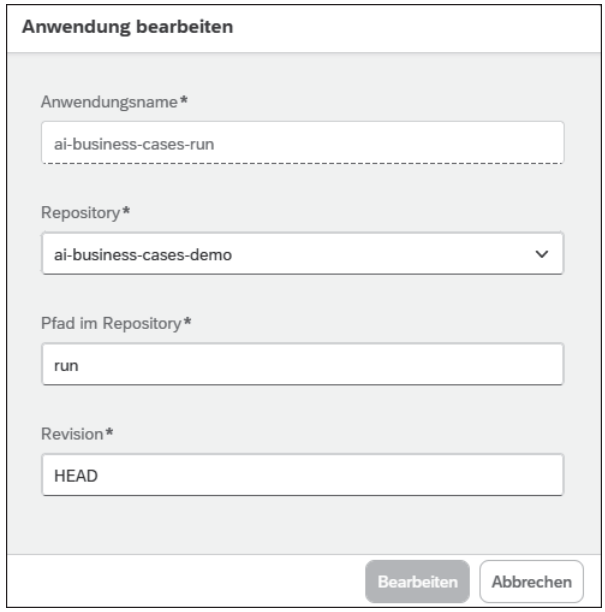

Abbildung 4.8 Anwendung in SAP AI Core registrieren

Diese Anwendung dient anschließend als Grundlage für Trainings-, Deployment- oder Serving-Prozesse – je nachdem, welche Workflows und Codedateien sich im angegebenen Pfad befinden.

4.4.3 Szenario konfigurieren

Definition: Szenario

Nach erfolgreicher Konfiguration und Synchronisation der Anwendung durch den SAP AI Core stehen die Inhalte nun für operative Prozesse bereit. Unter **ML Operations • Szenarios** sind alle registrierten Anwendungen in Form sogenannter *Szenarien* sichtbar (siehe Abbildung 4.9). Ein Szenario repräsentiert dabei eine Einheit, mit der ein konkreter Trainings-, Serving- oder Ausführungsprozess initiiert werden kann.

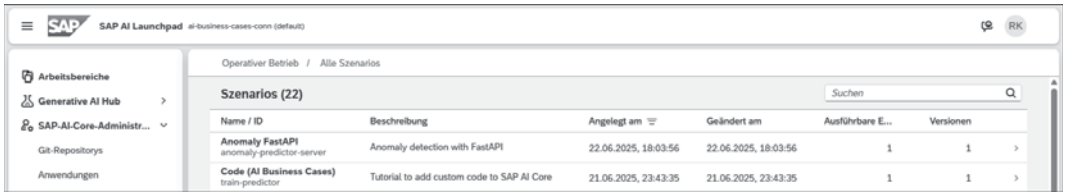

Abbildung 4.9 Übersicht der verfügbaren Szenarien im SAP AI Launchpad

Details der Szenarien

Die Szenarien sind jeweils mit einem Namen, einer Beschreibung, dem Erstellungsdatum sowie den verfügbaren Versionen versehen. Sie bieten den zentralen Einstiegspunkt für den weiteren Betrieb der KI-Anwendung im SAP AI Launchpad.

Konfiguration erstellen

Um ein registriertes Szenario tatsächlich ausführen zu können – sei es ein Training, ein Inferenzserver oder eine Vorhersage –, müssen Sie im nächsten Schritt eine sogenannte *Konfiguration* erstellen. Die Konfiguration enthält alle notwendigen Angaben darüber, welche Einheit (z. B. ein Skript oder Container) in welchem Szenario ausgeführt werden soll.

Dieser Schritt erfolgt über den Menüpunkt **ML Operations • Konfigurationen** (siehe Abbildung 4.10). Mit einem Klick auf **Anlegen** öffnet sich ein Dialog, in dem Sie eine bestehende Einheit (aus dem Szenario) auswählen, benennen und bei Bedarf mit weiteren Parametern versehen können.

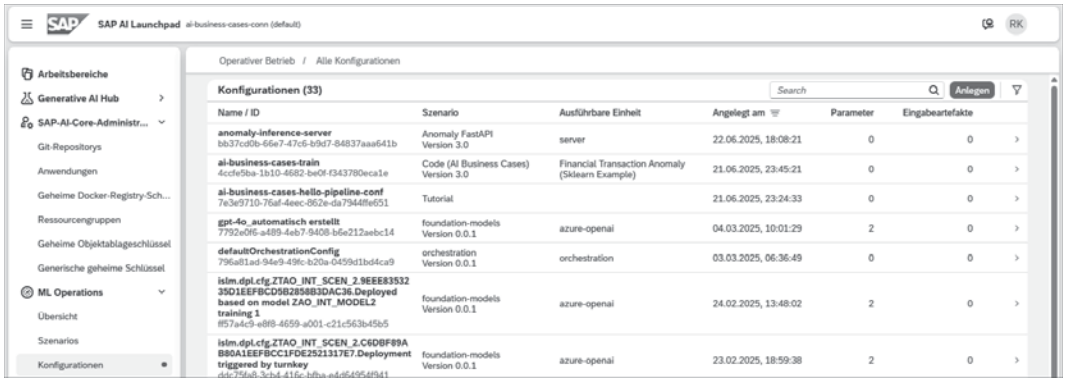

Abbildung 4.10 Übersicht über Konfigurationen im SAP AI Launchpad

Die Konfiguration ist damit der konkrete Startpunkt, um z. B. das Training des Anomalieerkennungsmodells oder den Betrieb eines FastAPI-basierten Inferenzservers auszulösen.

Konfigurations-Wizard

Nachdem Sie den Konfigurationsprozess gestartet haben, führt Sie ein Wizard Schritt für Schritt durch die benötigten Einstellungen (siehe Abbildung 4.11). Im ersten Schritt wird die neue Konfiguration benannt und das gewünschte **Szenario**, die entsprechende **Version** sowie die **Ausführbare Einheit** (z. B. ein Trainingsskript oder Vorhersagemodell) festgelegt.

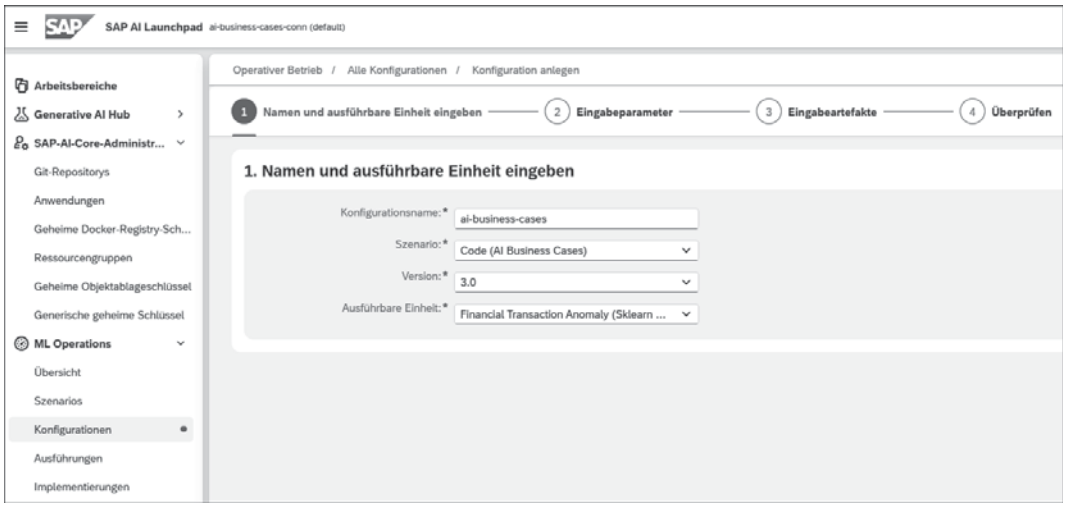

Abbildung 4.11 Konfigurations-Wizard

Angaben überprüfen

In den weiteren Schritten des Wizards können Sie bei Bedarf zusätzliche **Eingabeparameter** (z. B. Schwellenwerte oder Flags) und **Eingabeartefakte** (z. B. Dateien, Modelle oder Datensätze) definieren. Am Ende erfolgt eine Überprüfung aller Angaben, bevor Sie die Konfiguration erstellen und aktivieren.

Damit ist der Weg zur automatisierten Ausführung Ihrer individuellen KI-Pipeline in SAP AI Core geebnet.

4.4.4 Workflow ausführen

Ausführungs-Wizard

Nachdem Sie die Konfiguration erfolgreich abgeschlossen haben, kann die eigentliche Ausführung des Szenarios gestartet werden – z. B. für das initiale Training eines KI-Modells. Dies erfolgt im Bereich **ML Operations • Ausführungen**. Durch einen Klick auf **Anlegen** startet der Ausführungs-Wizard, in dem Sie das entsprechende Szenario auswählen und die gewünschte ausführbare Einheit (z. B. ein Trainingsskript) festlegen (siehe Abbildung 4.12).

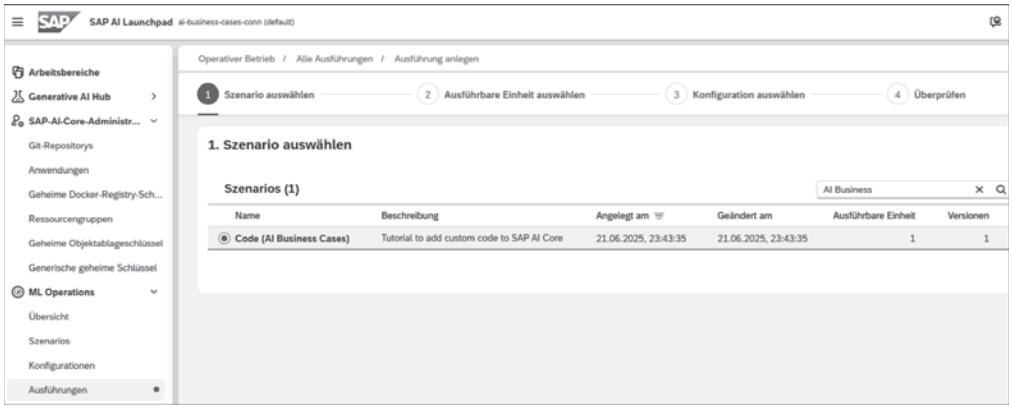

Abbildung 4.12 Ausführungs-Wizard

Optional: Ausführung planen

Optional können Sie die Ausführung auch planen – etwa zur regelmäßigen Aktualisierung des Modells auf Basis neuer Daten. Dies eröffnet die Möglichkeit, Machine-Learning-Workflows vollständig automatisiert in der SAP BTP zu betreiben.

Status überwachen

Nach dem Start einer Ausführung lässt sich der Fortschritt direkt im SAP AI Launchpad nachvollziehen. Unter **ML Operations • Ausführungen** kann jede gestartete Instanz anhand ihrer ID und ihres aktuellen Status überwacht werden – z. B. ob sie gerade läuft, abgeschlossen oder fehlgeschlagen ist.

Neben dem visuellen Prozessdiagramm sind dort auch Start-/Endzeiten, Dauer, Logs und etwaige Ergebnisse abrufbar (siehe Abbildung 4.13).

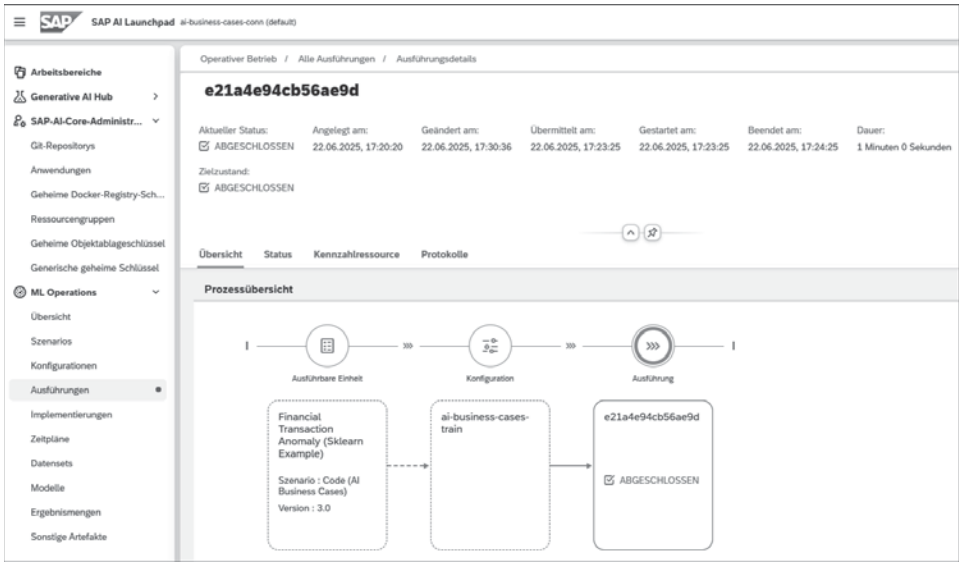

Abbildung 4.13 Prozessübersicht: Ablauf der Ausführung

Das erleichtert sowohl die Fehlersuche als auch die Nachverfolgung wiederkehrender Trainings- oder Inferenzprozesse.

4.4.5 Modell implementieren

Handelt es sich nicht um eine einmalige Ausführung wie beim Modelltraining, sondern um ein dauerhaft erreichbares Modell (z. B. via REST-API), erfolgt die Bereitstellung über **ML Operations • Implementierungen**.

Neues Deployment erstellen

Über **Anlegen** können Sie ein neues Deployment erstellen. Dabei wird das gewünschte, zuvor synchronisierte Serving-Template ausgewählt – in unserem Fall das Modell zur Anomalieerkennung mit FastAPI, »Anomaly FastAPI« (siehe Abbildung 4.14). Es wird anschließend dauerhaft bereitgestellt und ist für Anfragen von außen erreichbar.

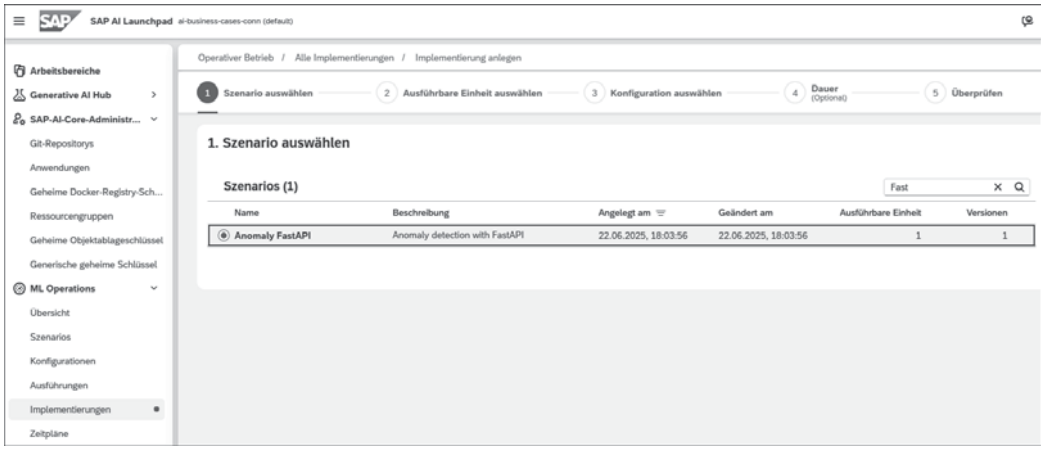

Abbildung 4.14 Synchronisiertes Serving-Template auswählen

Bereitstellungseinheit auswählen

Nachdem das Szenario gewählt wurde, erfolgt im zweiten Schritt die Auswahl der ausführbaren Bereitstellungseinheit. Diese Einheit definiert den konkreten Bereitstellungsbaustein – also zum Beispiel eine API, die später Anfragen entgegennimmt und Vorhersagen trifft. In unserem Fall wird die Einheit `server` verwendet, die unsere FastAPI-Anwendung mit unserem Isolation-Forest-Modell bereitstellt (siehe Abbildung 4.15).

Konfiguration auswählen

Im dritten Schritt wählen Sie eine bestehende Konfiguration aus, die zur gewählten ausführbaren Bereitstellungseinheit gehört. Diese Konfiguration legt fest, wie das Modell beim Deployment verwendet wird – inklusive möglicher Parameter oder Eingabeartefakte. In unserem Fall ist die Konfiguration »anomaly-inference-server« bereits vorhanden und ausgewählt (siehe Abbildung 4.16). Alternativ können Sie an dieser Stelle auch eine neue Konfiguration erstellen.

Abbildung 4.15 Ausführbare Einheit auswählen

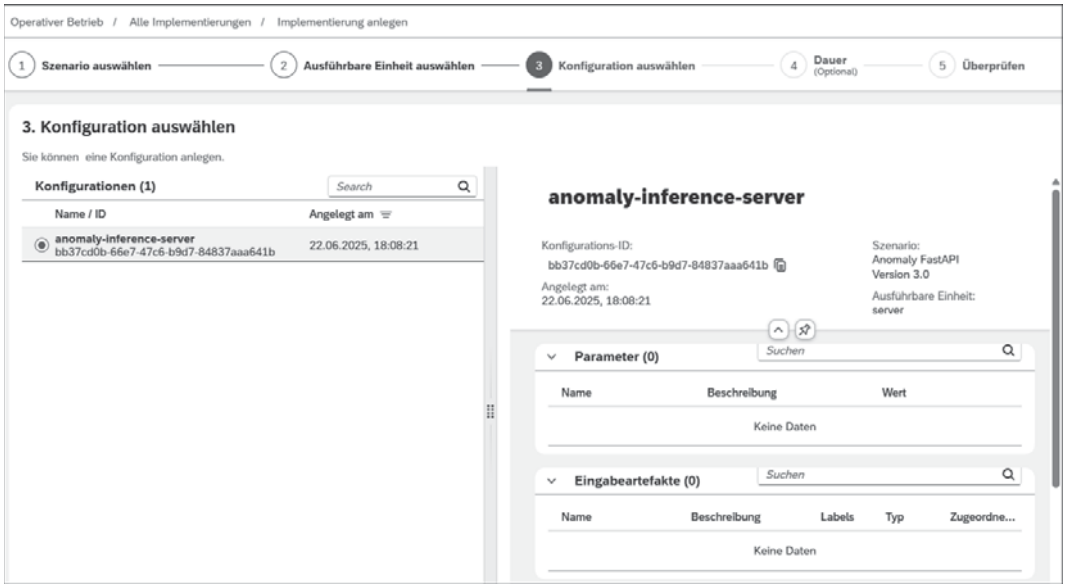

Abbildung 4.16 Eine bestehende Konfiguration auswählen

Dauer wählen

In nächsten Schritt des Deployment-Wizards kann die gewünschte **Dauer** der Bereitstellung gewählt werden. Dies ermöglicht entweder eine benutzerdefinierte Laufzeit (z. B. für Tests oder geplante Offline-Stellungen) oder eine Nutzung der Standarddauer, die systemseitig vordefiniert ist (siehe Abbildung 4.17). So wird gesteuert, wie lange das Modell aktiv im System verfügbar ist.

Einstellungen überprüfen

Im letzten Schritt des Wizards werden alle zuvor getroffenen Einstellungen zusammengefasst dargestellt: **Szenario**, **Ausführbare Einheit**, **Konfiguration** sowie die gewählte **Dauer** (siehe Abbildung 4.18). So lässt sich vor dem finalen Schritt noch einmal alles überprüfen und bei Bedarf korrigieren. Mit einem Klick auf **Anlegen** wird das Deployment in der SAP BTP angestoßen und automatisch ausgeführt.

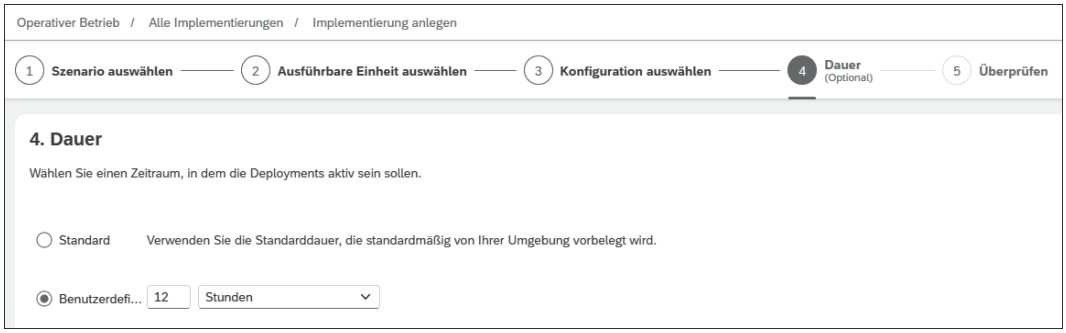

Abbildung 4.17 Dauer des Deployments festlegen

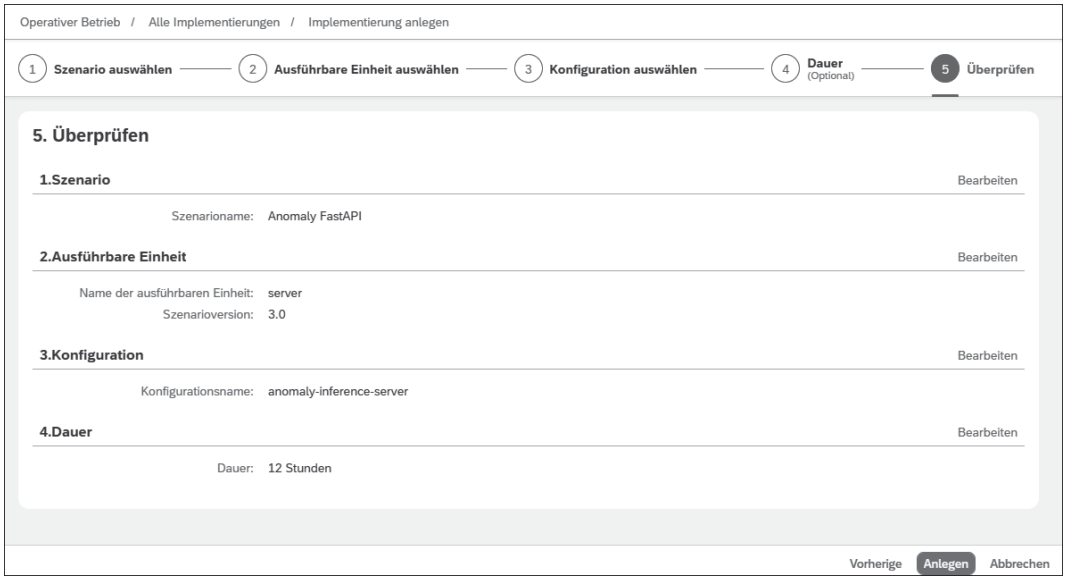

Abbildung 4.18 Deployment-Einstellungen überprüfen

Status prüfen

Nach der Ausführung kann der aktuelle Status des Deployments in der Detailansicht überprüft werden (siehe Abbildung 4.19). Es wird angezeigt, ob das Deployment bereits aktiv ist oder sich noch in Vorbereitung befindet. Sobald es vollständig aktiv ist, wird auch die zugehörige **URL** zur API-Nutzung angezeigt.

Sobald das Modell erfolgreich deployt wurde, wechselt der Status zu **WIRD AUSGEFÜHRT** (siehe Abbildung 4.20). In diesem Zustand ist das Modell aktiv und kann über die angegebene **URL** angesprochen werden. Damit ist der Service produktiv nutzbar – beispielsweise für Inferenzanfragen aus externen Anwendungen.

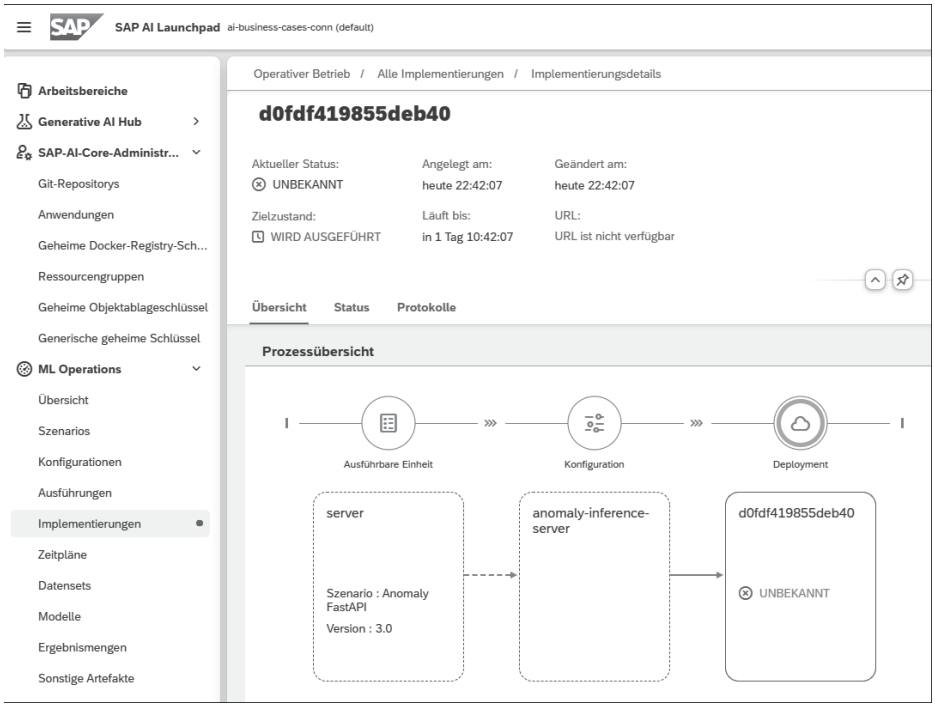

Abbildung 4.19 Status des Deployments

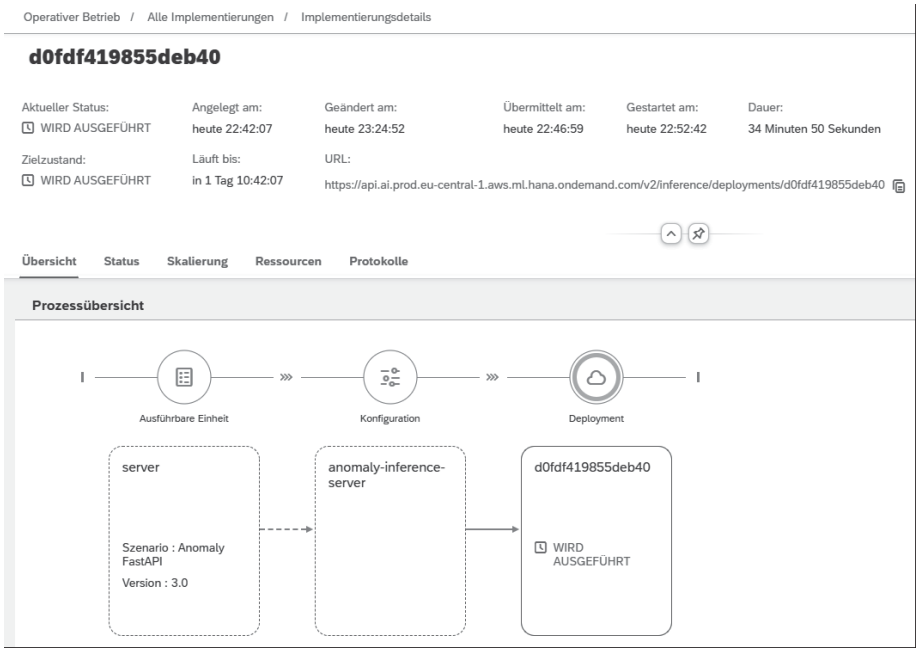

Abbildung 4.20 Aktiver Deployment-Status mit verfügbarer URL

4.4.6 Das Modell über eine API in neue oder bestehende Anwendungen integrieren

Nachdem das Modell erfolgreich deployt wurde und der Status auf **WIRD AUSGEFÜHRT** steht, kann der bereitgestellte Service nun programmgesteuert über eine REST-API angesprochen werden. Dies ermöglicht es, externe Anwendungen oder Testszenarien direkt mit dem Anomalieerkennungsmodell zu verbinden.

Voraussetzungen

Um die API nutzen zu können, müssen folgende Voraussetzungen erfüllt sein:

1. **Zugriffstoken (Access Token)**
   Für den Aufruf der geschützten API-Endpunkte ist ein gültiger *Bearer Token* erforderlich. Dieser kann mithilfe des zu Beginn erstellten Service Keys generiert werden – typischerweise über ein OAuth2-Flow mit Client Credentials.
2. **API-Endpunkt (URL)**
   Die vollständige URL zum Deployment wird in der Übersicht der SAP-AI-Launchpad-Deployment-Details angezeigt. Sie setzt sich aus der Basis-URL und dem spezifischen Pfad zusammen.

```
https://api.ai.prod.eu-central-1.aws.ml.hana.ondemand.com/v2/
inference/deployments/<deployment-id>/v2/predict
```

In unserem Fall lautet der Endpunkt:

```
https://api.ai.prod.eu-central-1.aws.ml.hana.ondemand.com/v2/
inference/deployments/d0fdf419855deb40/v2/predict
```

Beispiel-Request

Für die Kommunikation mit dem Service wird ein POST-Request an die oben genannte URL gesendet. Im Header wird der Access Token im Format `Authorization: Bearer <token>` übermittelt (siehe Listing 4.30).

```
POST /v2/inference/deployments/d0fdf419855deb40/v2/predict HTTP/1.1
Host: api.ai.prod.eu-central-1.aws.ml.hana.ondemand.com
Authorization: Bearer <ACCESS_TOKEN>
Content-Type: application/json
```

Listing 4.30 Request-Header

Body

Der Body enthält die Transaktionsdaten, die auf Anomalien überprüft werden sollen (siehe Listing 4.31).

```
{
    "data": [
        {
            "amount": 35.75,
            "hour": 15,
            "weekday": "Tue",
            "merchant_category": "Groceries"
        },
        {
            "amount": 10000.72,
            "hour": 10,
            "weekday": "Wed",
            "merchant_category": "Groceries"
        }
    ]
}
```

Listing 4.31 Request-Body (Beispiel)

API-Antwort

Die Antwort enthält für jede übergebene Transaktion eine Vorhersage (siehe Listing 4.32). Dabei signalisiert das Feld `anomaly_detected`, ob es sich bei der jeweiligen Transaktion um eine Anomalie handelt:

- `0`: Keine Anomalie erkannt
- `1`: Anomalie erkannt

```
{
    "predictions": [
        {
            "input": {
                "amount": 35.75,
                "hour": 15,
                "weekday": "Tue",
                "merchant_category": "Groceries"
            },
            "anomaly_detected": 0
        },
        {
            "input": {
                "amount": 10000.72,
                "hour": 10,
                "weekday": "Wed",
                "merchant_category": "Groceries"
            },
```

```
            "anomaly_detected": 1
        }
    ]
}
```

Listing 4.32 Beispielantwort

In diesem Beispiel wurde die erste Transaktion als normal erkannt, während die zweite Transaktion aufgrund des ungewöhnlich hohen Betrags als Anomalie klassifiziert wurde.

Nahtlose Integration möglich

Mit dem erfolgreich erstellten und getesteten Service steht nun ein voll funktionsfähiger Anomalieerkennungsdienst zur Verfügung, der über eine standardisierte REST-API angesprochen werden kann. Dadurch lässt sich der Service nahtlos in bestehende Applikationen, Workflows oder Backend-Systeme integrieren – sei es zur Überwachung von Transaktionen, zur Vorprüfung von Buchungen oder als Bestandteil eines größeren Analyseprozesses.

Da der Service auf der SAP BTP betrieben wird, profitiert die Einbindung zusätzlich von den Vorteilen einer skalierbaren, sicheren und cloud-nativen Infrastruktur.

Ob Webanwendung, mobile App oder automatisierter Job im Backend – der KI-Service kann an jeder Stelle eingebunden werden, an der Anomalien erkannt und weiterverarbeitet werden sollen.

Mit dem in diesem Kapitel entwickelten Anomalieerkennungsmodell auf Basis des Isolation Forests hat die SecureBank AG einen wichtigen Schritt in Richtung automatisierte Betrugserkennung gemacht. Durch die Bereitstellung des Modells als API steht eine flexible Schnittstelle zur Verfügung, die sich problemlos in bestehende Systeme integrieren lässt. Die Operationalisierung über GitHub und Docker sorgt dabei nicht nur für eine saubere Versionierung, sondern ermöglicht auch eine unkomplizierte Nutzung an verschiedenen Standorten. Dank der nahtlosen Einbindung in die SAP Business Technology Platform (SAP BTP) über den SAP AI Core ist eine vollständige Integration in die SAP-Infrastruktur der SecureBank AG gewährleistet.

Für die Mitarbeiter bedeutet das: Sie können sich künftig auf die gezielt als auffällig gemeldeten Transaktionen konzentrieren, anstatt jede einzelne Buchung manuell prüfen und überwachen zu müssen. Die Anomalieerkennung wird so zum immer aktiven »Service-Begleiter«, der verdächtige Vorgänge automatisch identifiziert und zur weiteren Überprüfung bereitstellt.

Damit ist die SecureBank AG in der Lage, Risiken effizienter zu managen und ihre Geschäftsprozesse nachhaltig zu optimieren.

4.5 Zusammenfassung

Moderne KI-gestützte Anomalieerkennung ist ein zentrales Werkzeug, um Betrug und Fehler in Transaktionen frühzeitig zu erkennen und Schäden wirksam zu verhindern. Durch die intelligente Kombination aus hochwertigen Daten, durchdachter Vorbereitung und dem gezielten Einsatz spezialisierter Machine-Learning- und Deep-Learning-Algorithmen wie Isolation Forests, Autoencodern oder Transformer-Modellen können auch in großen Datenmengen in Echtzeit verdächtige Muster zuverlässig identifiziert werden.

Der Weg von der lokalen Entwicklung über das Training und Deployment des Modells bis hin zur professionellen Integration als produktiver Cloud-Service (etwa mit SAP AI Core) zeigt, wie technische Exzellenz, Automatisierung und Skalierbarkeit in der Praxis Hand in Hand gehen. Die konsequente Einbindung von DevOps-Methoden, API-Schnittstellen und automatisierten Workflows garantiert nicht nur Effizienz und Reproduzierbarkeit, sondern auch die Einhaltung regulatorischer Vorgaben.

Das Anwendungsspektrum reicht dabei weit über die klassische Betrugserkennung hinaus und bietet auch in anderen Bereichen – von der internen Buchhaltung über die Logistik bis zur Produktion – einen echten Mehrwert. Entscheidend für den Erfolg sind stets eine klare Zieldefinition, eine enge Zusammenarbeit aller beteiligten Fachbereiche und die kontinuierliche Anpassung der Systeme an sich wandelnde Bedrohungen und Geschäftsprozesse.

Die automatisierte Anomalieerkennung mit KI ist heute ein wesentlicher Erfolgsfaktor für moderne Unternehmen. Sie schafft nicht nur Sicherheit und Vertrauen, sondern bildet auch die Grundlage für Innovation, Effizienz und nachhaltigen Geschäftserfolg.