Nadejda Ismailova · Anja Kammer
Larysa Visengeriyeva

MLOps

Nachhaltige Entwicklung und Betrieb von Machine-Learning-Anwendungen

INNOQ

MLOps

Nachhaltige Entwicklung und Betrieb von Machine-Learning-Anwendungen

Nadejda Ismailova
Anja Kammer
Dr. Larysa Visengeriyeva

innoQ Deutschland GmbH
Krischerstraße 100 · 40789 Monheim am Rhein · Germany
Phone +49 2173 33660 · www.INNOQ.com

Layout: Tammo van Lessen with XeLaTeX
Design: Sonja Scheungrab, Susanne Kayser
Print: mediaprint solutions GmbH, Paderborn, Germany

MLOps – Nachhaltige Entwicklung und Betrieb von Machine-Learning-Anwendungen
Published by innoQ Deutschland GmbH
1. Auflage · September 2022

Inhaltsverzeichnis

1 Einführung

1.1 Warum Machine Learning relevant ist?

Laut Statista Digital Economy Compass 2019[1] werden zwei folgende große Trends die Wirtschaft und unser Leben in der nächsten Zeit grundlegend verändern:

- Die datengetriebene Welt und die damit zusammenhängende exponentiell wachsende Menge an digital gesammelten Daten.
- Die zunehmende Bedeutung von Künstlicher Intelligenz, maschinellem Lernen und Data Science, die aus dieser enormen Menge an Daten Erkenntnisse ableitet.

Der Einfachheit halber werden wir im Folgenden den Begriff *Maschinelles Lernen (ML)* verwenden, die Konzepte gelten jedoch sowohl für die Bereiche Künstliche Intelligenz als auch Data Science.

Da ML ein leistungsfähiges Werkzeug ist, kann es viele praktische Probleme lösen. Ähnlich wie bei jedem anderen Software-Tool müssen wir den „richtigen" Nagel (Anwendungsfall oder Problem) identifizieren, um diesen „Hammer" (ML-Algorithmen) einzusetzen.

Wir sind daran interessiert, ML in Softwaresysteme einzubinden, weil ML einige Probleme lösen kann, die zu komplex sind, um auf traditionelle Weise gelöst zu werden. Für solche Probleme könnte eine probabilistische Lösung, die mit Hilfe von ML implementiert wird, der richtige Weg sein. Zum Beispiel können Perceptive Problems in Conversational UIs mit Techniken wie Spracherkennung oder Sentimentanalyse gelöst werden. ML scheint am besten geeignet zu sein, da solche Probleme eine große Anzahl von Elementen mit unterschiedlichen Repräsentationen haben. Eine andere Art von Problemen, die sich für ML eignen, sind Multiparameterprobleme. Zum Beispiel wenden wir Ansätze des ML an, um eine Vorhersage von Aktienkursen zu generieren, die eine Grundlage für eine Aktienhandelsentscheidung ist.

[1] https://cdn.statcdn.com/download/pdf/DigitalEconomyCompass2019.pdf

Aber was bedeutet es, ML in Softwaresysteme einzubinden? Jede Pipeline für ML ist ein Satz von Operationen, die ausgeführt werden, um ein Modell zu erzeugen. Ein ML-Modell ist grob definiert als eine mathematische Darstellung eines realen Prozesses. Wir können uns das ML-Modell als eine Funktion vorstellen, die einige Eingabedaten nimmt und eine Ausgabe erzeugt (Klassifizierung, Empfehlung oder Cluster). Die Leistung eines jeden Modells wird mit Hilfe von Bewertungsmetriken wie *Precision & Recall* oder *Accuracy* bewertet.

Modelle in die Produktion zu bringen bedeutet, sie den Softwaresystemen zur Verfügung zu stellen. In der Praxis können wir durch den Einsatz des ML-Modells die folgenden Funktionen bereitstellen (siehe auch „The AI Organization“ von David Carmona[2]):

- Empfehlung, die das relevante Produkt in einer großen Sammlung auf der Grundlage der Produktbeschreibung oder der vorherigen Interaktionen des Benutzers identifiziert.
- Auswahl von Top-K-Elementen, die eine Menge von Elementen in einer bestimmten Reihenfolge organisiert, die für den Benutzer geeignet ist (z. B. Suchergebnisse).
- Klassifizierung, die die Eingabebeispiele einer der zuvor definierten Klassen zuordnet (z. B. Spam/reguläre E-Mails).
- Vorhersage, die einer Entität von Interesse einen höchstwahrscheinlichen Wert zuordnet, z. B. den Wert einer Aktie.
- Inhaltegenerierung, bei der neue Inhalte durch Lernen aus bestehenden Beispielen erzeugt werden, z. B. die Fertigstellung einer Bach’schen Choralkantate durch Lernen aus seinen früheren Kompositionen.
- Question Answering, das eine explizite Frage beantwortet, z. B.: „Beschreibt dieser Text dieses Bild?‘
- Automatisierung, die eine Reihe von Benutzungsschritten automatisch ausführen kann, wie z. B. Aktienhandel.
- Betrugs- und Anomalie-Erkennung, um eine Aktion oder Transaktion als Betrug oder verdächtig zu identifizieren.

[2] https://www.oreilly.com/library/view/the-ai-organization/9781492057369/

- Informationsextraktion und -annotation, um wichtige Informationen in einem Text zu identifizieren, z. B. Namen von Personen, Stellenbeschreibungen, Unternehmen und Standorte.

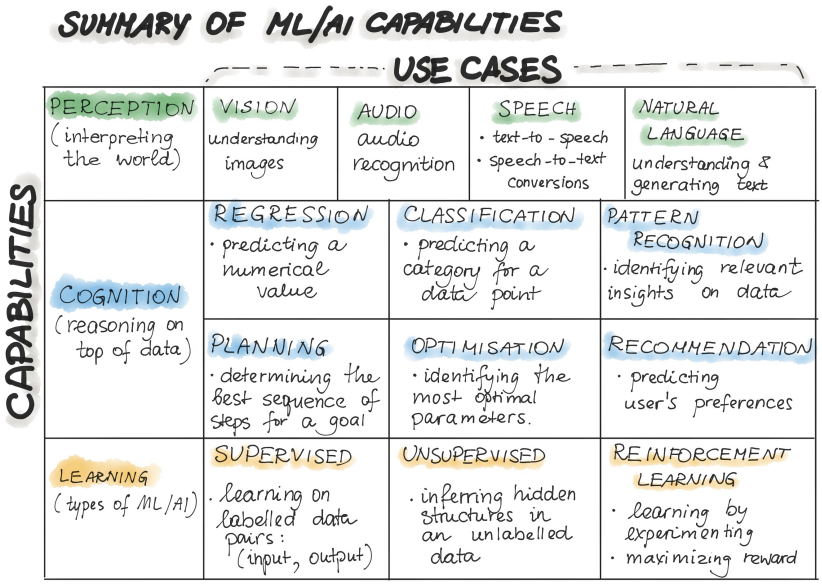

Abbildung 1.1: Zusammenfassung der möglichen Funktionen von ML (aus „The AI Organization“ von David Carmona)

1.2 Schwierigkeiten beim Deployment der ML-Modelle

Immer mehr Unternehmen experimentieren mit ML. Um ein Modell in die Welt zu bringen, ist allerdings mehr nötig, als es nur zu bauen. Um den vollen Nutzen aus dem erstellten ML-Modell zu ziehen, müssen wir es unserem Kernsoftwaresystem zur Verfügung stellen, indem wir das trainierte ML-Modell in die Kern-

Codebasis einbinden und in die Produktion einsetzen. Nach dem Deployment des Modells können andere Softwaresysteme dieses mit Daten versorgen und Vorhersagen erhalten, die wiederum in die Softwaresysteme zurückgeführt werden. Daher ist der volle Nutzen von ML-Modellen nur durch das ML-Modell-Deployment in der Produktion möglich.

Laut dem „2020 State of Enterprise Machine Learning - Bericht“ von Algorithmia[3] haben jedoch viele Unternehmen noch nicht herausgefunden, wie sie ihre ML-Ziele erreichen können. Denn die Überbrückung der Kluft zwischen der Erstellung von ML-Modellen und dem praktischen Einsatz ist immer noch eine anspruchsvolle Aufgabe. Es besteht ein grundlegender Unterschied zwischen der Erstellung eines ML-Modells im Jupyter-Notebook und dem Einsatz eines ML-Modells in einem Produktionssystem, das einen Mehrwert für das Unternehmen erzeugt. Obwohl die ML-Budgets steigen, haben nur 22 Prozent der Unternehmen[4], die ML einsetzen, ein ML-Modell erfolgreich in einer Produktionsumgebung eingesetzt.

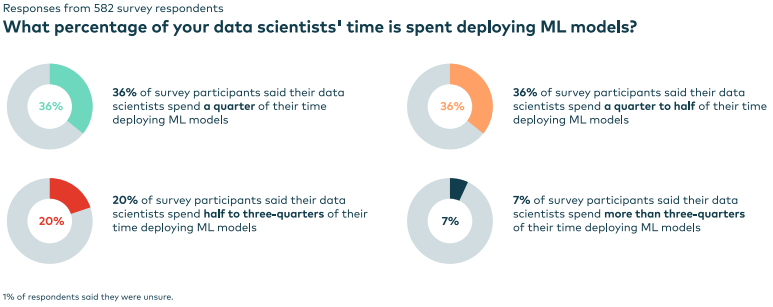

Abbildung 1.2: ML-Deployment-Schwierigkeiten (aus dem Bericht von Algorithmia, 2020)

[3] https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf

[4] https://designingforanalytics.com/resources/failure-rates-for-analytics-bi-iot-and-big-data-projects-85-yikes/

Der 2020 State of Enterprise Machine Learning Report basiert auf einer Umfrage unter fast 750 Personen, darunter ML-Fachleute, Manager von ML-Projekten und Führungskräfte in Tech-Unternehmen. Die Hälfte der Befragten antwortete, dass ihr Unternehmen zwischen einer Woche und drei Monaten braucht, um ein ML-Modell einzusetzen. Etwa 18 Prozent gaben an, dass es zwischen drei Monaten und einem Jahr dauert. Laut dem Bericht seien Skalierung, Versionskontrolle, Reproduzierbarkeit des Modells und die Abstimmung mit den Stakeholdern die größten Herausforderungen beim Einsatz von ML.

Der Grund für die zuvor beschriebenen Schwierigkeiten liegt darin, dass sich die Entwicklung der auf ML basierenden Anwendungen grundlegend von der Entwicklung herkömmlicher Software unterscheidet. Die Entwicklung einer ML-basierten Anwendung findet auf drei folgenden Ebenen statt: Daten, ML-Modell und Code. Es ist notwendig, auf allen drei Ebenen Änderungen und Anpassungen umzusetzen. Das bedeutet, dass bei Systemen, die auf ML basieren, der Auslöser für einen Build die Kombination aus einer Code-, Daten- oder Modelländerung sein kann. Dies ist auch als „Changing Anything Changes Everything"-Prinzip[5] bekannt.

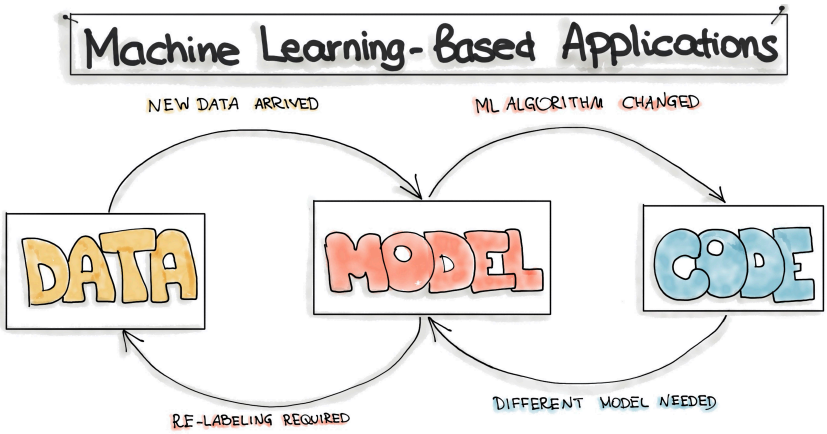

Abbildung 1.3: Drei Ebenen in ML Entwicklung

[5] https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Einige Szenarien für Änderungen in ML-Anwendungen sind z. B. die folgenden:

- Nach dem Einsatz des ML-Modells in einem Softwaresystem stellen wir vielleicht fest, dass das Modell mit der Zeit veraltet und sich abnormal verhält, so dass wir neue Daten benötigen, um unser ML-Modell neu zu trainieren.
- Nach der Untersuchung der verfügbaren Daten könnten wir erkennen, dass es schwierig ist, die Daten zu erhalten, die zur Lösung des zuvor definierten Problems benötigt werden, so dass wir das Problem neu formulieren müssten.
- In einigen Phasen des ML-Projekts könnten wir im Prozess zurückgehen und entweder mehr Daten sammeln, oder andere Daten sammeln und die Trainingsdaten neu auswählen. Dies sollte das erneute Training des ML-Modells auslösen.
- Nachdem wir das Modell den Endbenutzenden zur Verfügung gestellt haben, erkennen wir vielleicht, dass die Annahmen, die wir für das Training des Modells getroffen haben, falsch sind, so dass wir unser Modell ändern müssen.
- Manchmal kann sich das Geschäftsziel während der Projektentwicklung ändern und wir entscheiden uns, den ML-Algorithmus zum Trainieren des Modells anzupassen oder neu zu wählen.

Darüber hinaus gibt es drei häufige Probleme, die die Leistung von ML-Modellen beeinflussen, sobald sie in der Produktion sind.

Das erste Problem ist die **Datenqualität.** Da ML-Modelle aus Daten berechnet werden, sind sie empfindlich gegenüber der Semantik, der Menge und der Vollständigkeit der eingehenden Daten.

Das zweite Problem ist der **Modellverfall.** Die Leistung von ML-Modellen in der Produktion verschlechtert sich im Laufe der Zeit aufgrund von Änderungen in der realen Welt, in den realen Daten, die während des Modelltrainings nicht gesehen wurden, und dem Modell neu zugeführt werden müssen.

Das dritte Problem ist die **Lokalität.** Modelle, die auf unterschiedlichen Benutzungsdemografien vortrainiert wurden, können möglicherweise nicht auf neue Geschäftskundschaft oder andere Benutzungsgruppen übertragen werden und arbeiten für diese dann laut Qualitätsmetriken nicht korrekt.

Da ML immer weiter in neue Anwendungen expandiert und neue Branchen prägt, bleibt der Aufbau erfolgreicher ML-Projekte eine herausfordernde Aufgabe. Es besteht eine Notwendigkeit, nachhaltige und effektive Praktiken und Prozesse rund um das Design, die Erstellung und den Einsatz von ML-Modellen in der Produktion zu etablieren. Dies ist die Aufgabe von MLOps.

Zum Nachlesen: Why is DevOps for Machine Learning so different?[6]

[6] https://hackernoon.com/why-is-devops-for-machine-learning-so-different-384z32f1

2 MLOps - Definition

Wir haben die Probleme, die durch die Anwendung von ML gelöst werden könnten, und die Herausforderungen, die bei der Einführung von ML-Modellen in der Produktion entstehen, ermittelt. MLOps - oder Machine Learning Operations - soll uns dabei helfen, trotz dieser Herausforderungen die bestmögliche Leistung der eingesetzten ML-Modelle sicherzustellen und somit zuverlässig einen Mehrwert für das Geschäft und die Nutzerschaft durch den Einsatz von ML zu generieren.

Nun, was ist **MLOps?**

Die Interessengruppe MLOps SIG[1] der CD Foundation definiert den Begriff MLOps als „die Erweiterung der DevOps-Methodik, um ML und Data Science Assets als Elemente erster Klasse innerhalb der DevOps-Ökologie einzubeziehen".

Alternativ können wir die Definition des **Machine Learning Engineering** (**MLE**) von A. Burkov verwenden: „MLE verwendet wissenschaftliche Prinzipien, Tools und Techniken des maschinellen Lernens und des traditionellen Software-Engineerings zum Entwerfen und Erstellen komplexer Rechensysteme. MLE umfasst alle Stufen von der Datenerfassung über die Modellerstellung bis hin zur Bereitstellung des Modells für das Produkt oder die Verbraucher."

So wie DevOps, entstand MLOps aus der Erkenntnis, dass die Trennung der ML-Modellentwicklung von dem Prozess, der sie liefert - dem ML-Betrieb - die Qualität, Transparenz und Agilität der gesamten intelligenten Software senkt. Außerdem ist der Aufbau erfolgreicher ML-basierter Softwareprojekte an sich schwierig. Jede ML-basierte Software muss drei Hauptbestandteile verwalten: **Daten, Modell** und **Code.** Als eine DevOps-Erweiterung etabliert MLOps auf allen drei Ebenen effektive Praktiken und Prozesse rund um das Design, die Erstellung und den produktiven Einsatz von ML-Modellen.

Mit MLOps streben wir außerdem danach, technische Schulden in ML-Anwendungen zu vermeiden.

[1] https://github.com/cdfoundation/sig-mlops/blob/master/roadmap/2020/MLOpsRoadmap2020.md

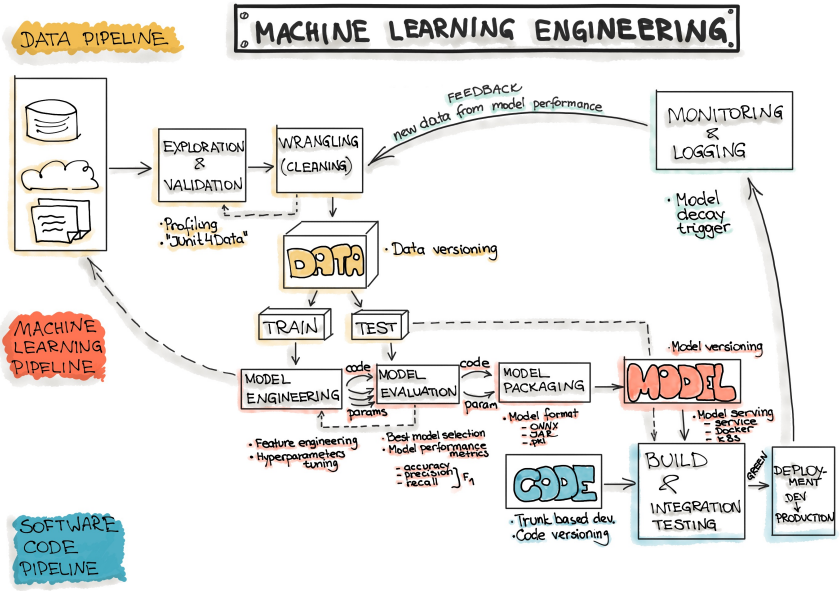

Abbildung 2.1: Machine-Learning-Engineering Überblick

MLOps SIG[2] definiert „*eine optimale MLOps-Erfahrung [als] eine, bei der ML-Assets konsistent mit allen anderen Software-Assets innerhalb einer CI/CD-Umgebung behandelt werden. ML-Modelle können zusammen mit den Services, die sie umhüllen, und den Services, die sie konsumieren, als Teil eines einheitlichen Release-Prozesses bereitgestellt werden.*“

Durch die Automatisierung dieser Praktiken soll die Einführung von ML in Softwaresystemen erleichtert werden und eine schnelle Bereitstellung von intelligenter Software ermöglicht werden.

In den folgenden Kapiteln werden wir die Phasen des *Data Engineering*, des *Model Engineering* und des *Software Release Engineering* durchgehen und wesentliche technische Methoden und Best Practices vorstellen, die in der Entwicklung und

[2] https://github.com/cdfoundation/sig-mlops/blob/master/roadmap/2020/MLOpsRoadmap2020.md

dem produktiven Betrieb der ML-basierten Software eingesetzt werden. Dabei gehen wir auf jede der drei Ebenen - Daten, Modell und Code - einzeln ein und setzen uns mit den spezifischen Aspekten dieser Ebenen auseinander. Anschließend beschreiben wir eine Reihe von Best Practices für die in MLOps wichtigen Konzepte, wie z. B. iterativ-inkrementelle Entwicklung, Automatisierung, Continuous Deployment, Versionierung, Testen, Reproduzierbarkeit und Monitoring.

3 Data Engineering Pipeline

Die Qualität der Datensätze hat einen großen Einfluss auf die Qualität und Performanz des ML-Modells und damit indirekt die Performanz des gesamten ML-basierten Softwaresystems. Das berühmte Zitat „Garbage In, Garbage Out“ bedeutet im Kontext von ML, dass das ML-Modell nur so gut sein wird, wie die im Training des ML-Modells verwendeten Daten. Die notwendige Menge und Qualität der Datensätze sind in der Regel problemspezifisch und können empirisch ermittelt werden. Das geschieht in der Phase des Data Engineering.

Data Engineering ist ein wichtiger Schritt und wird als sehr zeitaufwendig beschrieben. Die meiste Zeit eines ML-Projekts wird möglicherweise mit dem Aufbau von Datensätzen, der Bereinigung und der Transformation von Daten verbracht. Durch den Aufbau einer Data Engineering Pipeline und die Automatisierung der einzelnen Schritte kann dieser Prozess effizient gestaltet und wichtige Qualitätseigenschaften der Daten können reproduzierbar sichergestellt werden.

Eine Data Engineering Pipeline umfasst eine Abfolge von Operationen auf den verfügbaren Daten. Dazu gehören Data Ingestion, Exploration und Validierung, Data Wrangling (Bereinigung) und Data Splitting. Das Endziel dieser Operationen ist die Erstellung von Trainings- und Testdatensätzen für die ML-Algorithmen.

3.1 Data Ingestion

Data Ingestion (Datenaufnahme) ist das Sammeln von Daten mit Hilfe verschiedener Systeme, Frameworks und Formate, wie z. B. interne/externe Datenbanken, Data Marts, OLAP-Würfel, Data Warehouses, OLTP-Systeme, Spark, HDFS usw. Dieser Schritt kann auch die Generierung synthetischer Daten oder die Datenanreicherung beinhalten. Dieser Baustein der Data Engineering Pipeline umfasst folgende Jobs/Aktionen, die maximal automatisiert werden sollten:

- Identifizierung und Dokumentation der Datenquellen (Data Provenance).
- Schätzung und Beschaffung von ausreichendem Speicher für die Daten.
- Beschaffung und Formattierung der Daten, dabei soll es eifach sein mit dem Format zu arbeiten ohne die Daten selbst zu verändern.

- Sicherung einer Kopie der Daten, die Originaldatensätze müssen unangetastet bleiben.
- Einhaltung des Datenschutzes, d.h. es muss sichergestellt werden, dass sensible Daten gelöscht oder geschützt werden (z. B. anonymisiert), um die Einhaltung der DSGVO zu gewährleisten.
- Dokumentation der Metadaten zu den Datensätzen in einem Metadaten-Katalog, z. B. Größe, Format, Aliase, Zeitpunkt der letzten Änderung und Zugriffskontrolllisten (siehe z. B. auch Google's Goods Projekt)[1].
- Erstellung der Testdaten: Eine Stichprobe der Datensätze muss als Testdaten beiseitegelegt und darf auf keinen Fall im Training des Modells verwendet werden. Schauen Sie sich diese nicht an, um den „data snooping"-Bias zu vermeiden. Sie erkennen, ob Sie darauf hereingefallen sind, wenn Sie eine bestimmte Art von ML-Modell anhand des Testsatzes auswählen. Dies wird zu einer ML-Modellauswahl führen, die zu optimistisch ist und in Produktion nicht gut abschneiden wird.

3.2 Exploration und Validierung

Das Ziel der Exploration ist die Erstellung von Datenprofilen, um Informationen über den Inhalt und die Struktur der Daten zu erhalten. Das Ergebnis dieses Schritts ist ein Satz von Metadaten, wie z. B. Maximum, Minimum oder Durchschnitt von Werten. Die Validierung ist ein Prozess, bei dem die Qualität der Daten durch Ausführen von Datenvalidierungsroutinen beurteilt wird. Datenvalidierungsoperationen sind fachlich definierte Fehlererkennungsfunktionen, die die Datensätze scannen, um Fehler zu erkennen. Sind z. B. bei Adressattributen die einzelnen Komponenten konsistent? Ist die richtige Postleitzahl mit der Adresse verknüpft? Gibt es fehlende Werte in den relevanten Attributen? Beim Aufbau dieses Schritts in der Data Engineering Pipeline sollten folgende Maßnahmen berücksichtigt werden:

- Verwendung von Tools, die Rapid Application Development und Dokumentation über die Datenexploration und Experimente unterstützen, z. B. Jupyter-Notebooks.

[1] https://dl.acm.org/doi/pdf/10.1145/2882903.2903730

- Attribut-Profiling, d.h. Erstellung und Dokumentation von Metadaten zu jedem Attribut, wie z. B. Anzahl der Datensätze, Datentyp (kategorisch, numerisch, Text, usw.), statistische Merkmale (min, max, median etc.), „Missing Value Ratio“ (also Anzahl der fehlenden Werte geteilt durch Anzahl der Datensätze), Art der Verteilung (Gauß, gleichmäßig, logarithmisch, etc.).
- Identifizierung der Label-Attribute für den Einsatz von Supervised Learning.
- Datenvisualisierung durch Darstellung der Werteverteilung.
- Analyse der Korrelationen zwischen Attributen.
- Identifikation von weiteren Daten, die für die Erstellung des Modells nützlich wären.

3.3 Data Wrangling

Data Wrangling (Bereinigung) ist ein Schritt der Datenvorbereitung, bei dem Daten programmatisch bearbeitet werden, z. B. durch Umformatierung oder Umstrukturierung bestimmter Attribute. Das Datenschema könnte dabei verändert werden. Alle Datentransformationen müssen durch Skripte und Funktionen in der Data Engineering Pipeline vorgenommen werden, damit auch zukünftige Daten denselben Prozess durchlaufen. Datenbereinigungsteil der Data Engineering Pipeline kann aus folgenden Transformationen bestehen:

- Korrigieren oder Entfernen der Ausreißer.
- Auffüllen der fehlenden Werte (z. B. mit Null, Mittelwert, Median) oder Entfernen der betroffenen Datensätze.
- Entfernen der Daten, die für das spätere Feature Engineering nicht relevant sind.
- Umstrukturieren der Daten, z.B Verschieben von Spalten, Erstellen neuer Spalten durch Extrahieren von Werten oder Kombinieren mehrerer Datensatzfelder, Filtern, Aggregationen etc.

Zum Nachlesen: „Principles of Data Wrangling“[2]

[2] https://learning.oreilly.com/library/view/principles-of-data/9781491938911/

3.4 Datenaufteilung

Nach der Bereinigung der Daten kommt die Datenaufteilung in Trainings-, Validierungs- und Testdatensätze. Trainingsdatensätze umfassen dabei etwa 80 Prozent der gesamten Daten.

Die Data Engineering Pipeline stellt die Trainings-, Validierungs- und Testdatensätze nun der Model Training Pipeline bereit.

4 Model Training Pipeline

Der Kern des ML-Workflows ist die Findungsphase der ML-Algorithmen fürs Model Training. Diese Phase wird von einer Model Training Pipeline unterstützt. Die Model Training Pipeline wird normalerweise von einem Data-Science-Team verwendet und umfasst eine Reihe von Vorgängen, die zu einem endgültigen Modell führen. Zu diesen Vorgängen gehören Model Training, Auswertung, Tests und Paketierung. Alle Vorgänge sollten so weit wie möglich automatisiert werden, damit das Data-Science-Team seine wertvolle Zeit der Analyse und dem Design der Algorithmen widmen kann.

4.1 Model Training

Model Training ist der Prozess der Anwendung eines ML-Algorithmus auf Trainingsdaten, um ein ML-Modell zu erhalten. Der Model-Training-Abschnitt in der Model Training Pipeline umfasst zwei große Bausteine: Feature Engineering und Model Engineering. Feature Engineering und Model Engineering sind umfangreiche Prozesse mit vielen Zwischenschritten. Zur möglichen Vorgehensweise im Feature Engineering und Model Engineering verweisen wir an dieser Stelle auf „Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow“ von Aurélien Géron[1] und listen hier nur eine Auswahl an Zwischenschritten auf.

Der Feature-Engineering-Teil der Pipeline könnte folgende Aufgaben umfassen:

- Umwandeln der kontinuierlichen Features in diskrete Werte.
- Zerlegen von Features (kategorisch, Datum/Zeit etc.).
- Hinzufügen von Transformationen von Features (log(x), sqrt(x), x2, usw.).
- Aggregieren von Features zu vielversprechenden neuen Features.
- Standardisieren oder Normalisieren von Features.

Die bereitgestellten Features werden nun im Model-Engineering-Teil der Pipeline verwertet. Der Arbeitsablauf könnte da wie folgt aussehen:

[1] https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/app02.html#project_checklist_appendix

- Jede ML-Modellspezifikation (Code, der ein ML-Modell erstellt) sollte ein Code-Review durchlaufen und versioniert werden.
- Es werden mehrere ML-Algorithmen aus verschiedenen Kategorien (z. B. lineare Regression, logistische Regression, k-means, naive Bayes, SVM, Random Forest) mit Standardparametern angewandt.
- Die Leistung der resultierenden Modelle wird gemessen und verglichen, z.B durch eine n-fache Kreuzvalidierung.
- Analyse der Fehlertypen, die die ML-Modelle machen.
- Eventuell weitere Feature-Auswahl und -Entwicklung.
- Auswahl von drei bis fünf vielversprechendsten Modellen mit unterschiedlichen Fehlertypen.
- Abstimmung der Hyperparameter durch Verwendung von Kreuzvalidierung.
- Evaluation der Ensemble-Methoden wie Majority Vote, Bagging, Boosting oder Stacking, um eine bessere Leistung zu erzielen (siehe auch „Ensemble Methods: Foundations and Algorithms“ von Zhi-Hua Zhou[2]).

Zum Nachlesen: „Feature Engineering for Machine Learning. Principles and Techniques for Data Scientists“ von Alice Zheng, Amanda Casari[3]

4.2 Modellauswertung und Modelltest

Nach dem Model Training stehen Modellauswertung und Modelltest an.

Bei der Modellauswertung wird das ausgewählte Modell validiert, um sicherzustellen, dass es die ursprünglichen Geschäftsziele erfüllt.

Ein Modelltest wird auf Testdaten durchgeführt. Dabei wird die Leistung gemessen und der Generalisierungsfehler geschätzt. Außerdem wird der endgültige Model Acceptance Test durchgeführt. Weiteres zu Tests wird im Kapitel „Best Practices“ behandelt.

[2] https://www.amazon.com/exec/obidos/ASIN/1439830037/acmorg-20
[3] http://shop.oreilly.com/product/0636920049081.do

4.3 Modellpaketierung

Modellpaketierung ist der Exportprozess in ein bestimmtes Format (z. B. PMML, PFA oder ONNX) damit die Anwendung das ML-Modell verwenden kann. In diesem Format kann das Modell dann an die konsumierenden Systeme übergeben werden. Es gibt verschiedene Formate zur Verteilung von ML-Modellen. Um ein verteilbares Format zu erreichen, sollte das ML-Modell als eigenständiges Asset ausführbar sein. Zum Beispiel könnte man ein Scikit-learn-Modell in einem Spark-Job verwenden wollen. Das bedeutet, dass die ML-Modelle außerhalb der Trainingsumgebung funktionieren sollten. Es gibt sprachenunabhängige und herstellerspezifische Austauschformate für ML-Modelle. Die folgende Tabelle fasst alle ML-Modell-Serialisierungsformate zusammen:

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-readable	Compression
"Amalgamation"	–	–	–	–	–	–	✓
PMML	✓	DMG	.pmml	AGPL	R, Python, Spark	✓ (XML)	✘
PFA	✓	DMG	JSON		PFA-enabled runtime	✓ (JSON)	✘
ONNX	✓	SIG LFAI	.onnx		TF, CNTK, Core ML, MXNet, ML.NET	–	✓
TF Serving Format	✓	Google	.pf		Tensor Flow	✘	g-zip
Pickle Format	✓		.pkl		scikit-learn	✘	g-zip
JAR/ POJO	✓		.jar		H2O	✘	✓
HDF	✓		.h5		Keras	✘	✓
MLEAP	✓		.jar/ .zip		Spark, TF, scikit-learn	✘	g-zip
Torch Script	✘		.pt		PyTorch	✘	✓
Apple .mlmodel	✘	Apple	.mlmodel		TensorFlow, scikit-learn, Core ML	–	✓

Abbildung 4.1: ML-Modell-Serialisierungsformate

5 Deployment Pipeline

Nachdem uns nun ein trainiertes, getestetes, validiertes Modell als Package bereitsteht, kann dieses im Produktivsystem eingesetzt werden. Bei der Integration des ML-Modells in die Service-Landschaft des Produktionssystems müssen folgende Prozesse etabliert werden:

1. Modell-Serving - der Prozess des Deployments des ML-Modells in eine Produktionsumgebung.
2. Modell-Monitoring - der Prozess des Beobachtens der Modell-Performanz in einer produktiven Umgebung auf der Basis von zuvor ungesehenen Daten. Insbesondere sind wir an ML-spezifischen Metriken interessiert, wie z. B. der Abweichung der Vorhersage von der vorherigen Modellleistung. Diese Metriken können als Auslöser für ein erneutes Model Training verwendet werden.
3. Logging - alle Anfragen ans ML-Modell sollten in einem Log festgehalten werden.

Wir behandeln Monitoring und Logging näher im Kapitel „Best Practices“.

Hier gehen wir als Nächstes auf die Gestaltung der Bereitstellung und des Betriebs einer ML-Anwendung in einer produktiven Umgebung ein. Dabei spielen die folgenden verschiedenen Aspekte eine Rolle:

- Typ des Modells und daraus ableitend die Architekturform des ML-Workflows.
- Modell-Serving-Muster aus der Sicht des konsumierenden Systems.
- Bereitstellungsstrategien aus infrastruktureller Sicht.

5.1 Verschiedene Architekturformen von ML-Workflows

Die Bereitstellung und der Betrieb eines ML-Modells in einer Produktionsumgebung kann abhängig vom Typ des Modells verschiedene Formen annehmen. Als Nächstes stellen wir hier vier mögliche Architekturformen vor. Diese können entlang der zwei folgenden Dimensionen klassifiziert werden:

1. Typ des Modell-Trainings.

2. Typ der Vorhersage.

Im Model Training wird zwischen folgenden Ansätzen unterschieden:

- Offline-Lernen (auch Batch- oder statisches Lernen genannt): Das Modell wird auf einem Satz von bereits gesammelten Daten trainiert. Nach dem Deployment in der Produktionsumgebung bleibt das ML-Modell gleich, bis es wegen Veralterung erneut trainiert werden muss. Dieses Phänomen der Veralterung eines Modells wird als „Modellverfall" (*engl. Model Decay*) bezeichnet und sollte sorgfältig überwacht werden.
- Online-Lernen (auch dynamisches Lernen genannt): Das Modell wird kontinuierlich neu trainiert, wenn neue Daten, z. B. in Form von Datenströmen, eintreffen. Dies ist in der Regel bei ML-Systemen der Fall, die Zeitreihendaten verwenden (z. B. Sensor- oder Aktienhandelsdaten), um die zeitlichen Effekte im ML-Modell zu berücksichtigen.

Die Vorhersagen können in zwei verschiedenen Weisen erstellt werden:

- Im Falle von Batch-Vorhersagen macht das eingesetzte ML-Modell eine Reihe von Vorhersagen, die auf historischen Eingabedaten basieren. Dies ist oft ausreichend für Daten, die nicht zeitabhängig sind, oder wenn es nicht entscheidend ist, Echtzeit-Vorhersagen als Ausgabe zu erhalten.
- Echtzeit-Vorhersagen (auch On-Demand-Vorhersagen genannt) werden in Echtzeit anhand der Eingabedaten generiert, die zum Zeitpunkt der Erstellung der Vorhersage verfügbar sind.

Nachdem wir diese beiden Dimensionen identifiziert haben, können wir die Bereitstellung von ML-Modellen in einer Produktionsumgebung in folgende vier Architekturformen einteilen: Forecast, Web-Service, Online Learning und AutoML.

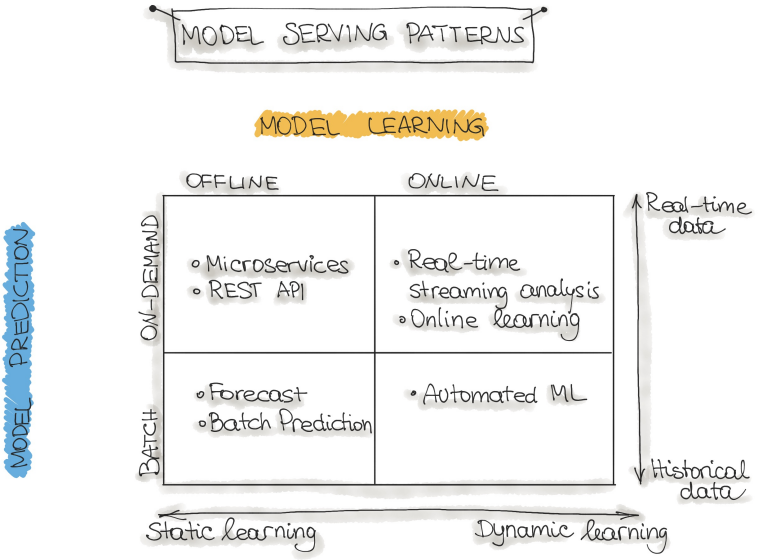

Abbildung 5.1: ML-Workflows

Forecast (Batch-Vorhersagen)

Diese Art von ML-Workflow wird verwendet, um mit ML-Algorithmen und Daten zu experimentieren, da es der einfachste Weg ist, ein ML-System zu erstellen. Normalerweise nehmen wir einen verfügbaren Datensatz, trainieren das ML-Modell, lassen dieses Modell dann auf anderen (meist historischen) Daten laufen, und das ML-Modell macht Vorhersagen. Dieser ML-Workflow ist in einem industriellen Umfeld und für Produktionssysteme nicht sehr nützlich und nicht üblich.

Web-Dienst

Die am häufigsten beschriebene Deployment-Architektur für ML-Modelle ist ein Web-Service (z. B. als Microservice). Der Web-Service nimmt Eingabedaten entgegen und gibt eine Vorhersage für die Eingabedatenpunkte aus. Das Modell wird offline auf historischen Daten trainiert, aber es verwendet reale Daten, um

Vorhersagen zu treffen. Der Unterschied zum Forecast besteht darin, dass das ML-Modell nahezu in Echtzeit für jeden einzelnen Datenpunkt ausgeführt wird. Der Web-Service verwendet Echtzeitdaten, um Vorhersagen zu treffen, aber das Modell bleibt konstant unverändert, bis es neu trainiert und wieder dem Produktionssystem bereitgestellt wird.

Die folgende Abbildung veranschaulicht die Architektur für das Integrieren trainierter Modelle als einsatzfähige Services. Die Methoden für die Integration werden wir im Abschnitt „Bereitstellungsstrategien“ besprechen.

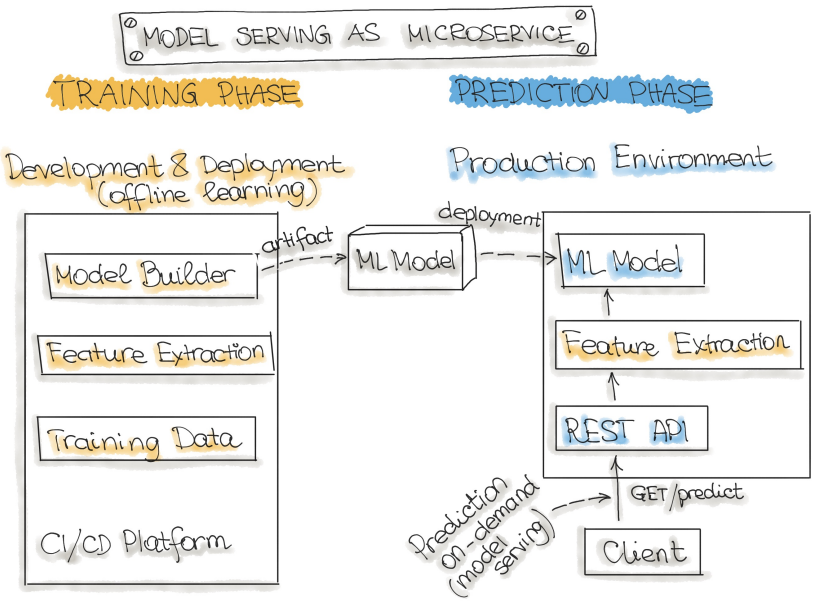

Abbildung 5.2: Web Service Pattern

Online-Lernen

Die dynamischste Art, ML in ein Produktionssystem einzubetten, ist die Implementierung von Online-Lernen.

Bei diesem Workflow erhält der ML-Algorithmus kontinuierlich einen Datenstrom, entweder als einzelne Datenpunkte oder in kleinen Gruppen (Mini-Batches). Das System lernt über neue Daten, sobald diese eintreffen, so dass das ML-Modell kontinuierlich schrittweise mit neuen Daten neu trainiert wird. Dieses ständig neu trainierte Modell ist sofort als Web-Service verfügbar.

Technisch gesehen funktioniert diese Art von ML-System gut mit der Lambda-Architektur in Big-Data-Systemen. Normalerweise sind die Eingabedaten ein Event-Stream, und das ML-Modell nimmt die Daten, wenn sie in das System eingehen, liefert Vorhersagen und lernt auf diesen neuen Daten neu. Das Modell wird typischerweise als Service auf einem Kubernetes-Cluster oder Ähnlichem ausgeführt.

Eine große Schwierigkeit mit dem Online-Lernsystem in Produktion besteht darin, dass das ML-Modell sowie die gesamte Systemleistung zunehmend abnehmen, wenn schlechte Daten in das System gelangen.

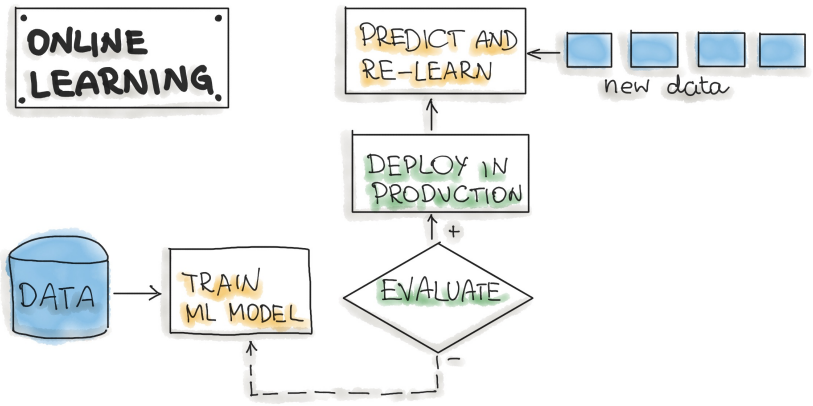

Abbildung 5.3: Online Learning ML-System (aus „Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow“, Kapitel 1 „The Machine Learning Landscape“, von Aurlien Gron)

AutoML

Eine noch anspruchsvollere Version des Online-Lernens ist das automatisierte ML oder AutoML.

AutoML erhält viel Aufmerksamkeit und wird als nächster Schritt in Enterprise ML angesehen. AutoML verspricht das Trainieren von ML-Modellen mit minimalem Aufwand und ohne Fachwissen über ML. Nutzende müssen Daten bereitstellen und das AutoML-System wählt automatisch einen ML-Algorithmus aus, z. B. eine neuronale Netzwerkarchitektur, und konfiguriert den ausgewählten Algorithmus. Anstatt das Modell zu aktualisieren, wird eine komplette Modell-Training-Pipeline in der Produktionsumgebung ausgeführt, die „on the fly" zu neuen Modellen führt.

Im Moment ist dies eine sehr experimentelle Art, ML-Workflows zu implementieren. AutoML wird normalerweise von großen Cloud-Anbietern wie Google[1] oder MS Azure[2] bereitgestellt. Allerdings müssen die mit AutoML erstellten Modelle die für den Erfolg in der Praxis erforderliche Genauigkeit erreichen.

Zum Nachlesen:

- AutoML: Überblick und Tools[3]
- AutoML-Benchmark[4]

5.2 Muster für Modell-Serving

Aus der Sicht des konsumierenden Systems kann beim Modell-Serving zwischen fünf Ansätzen, beziehungsweise Mustern, unterschieden werden: Model as Service, Model as Dependency, Precompute, Model on Demand und Hybrid Serving.

[1] https://cloud.google.com/automl/
[2] https://docs.microsoft.com/en-us/azure/machine-learning/concept-automated-ml
[3] https://www.automl.org/automl/
[4] https://www.researchgate.net/profile/Marc_Andre_Zoeller/publication/332750780_Benchmark_and_Survey_of_Automated_Machine_Learning_Frameworks/links/5e15bd1792851c8364baa47a/Benchmark-and-Survey-of-Automated-Machine-Learning-Frameworks.pdf

Diese unterscheiden sich in der Art, wie das konsumierende System auf das ML-Modell zugreifen kann und ob das konsumierende System und das ML-Modell zusammen ein Service bilden oder unabhängig bleiben.

ML-Modell-Serving-Taxonomie			
	ML-Modell		
Service & Versionierung	Zusammen mit der konsumierenden Anwendung	Unabhängig von der konsumierenden Anwendung	
Verfügbarkeit	Verfügbar während des Kompilierens und der Laufzeit	Verfügbar über eine REST API oder RPC	Verfügbar während der Laufzeit
Serving Patterns	"Model as Dependency"	"Model as Service"	"Precompute", "Model on Demand"
	Hybrid Model Serving (Federated Learning)		

Abbildung 5.4: Model-Serving-Muster

Model as Service

Model as Service ist ein gängiges Muster für die Umhüllung eines ML-Modells als unabhängiger Service. Wir können das ML-Modell und den Interpreter in einen dedizierten Webdienst verpacken, den die Anwendungen über eine REST-API anfordern oder als RPC-Dienst konsumieren können.

Dieses Muster kann für verschiedene ML-Workflows verwendet werden, z. B. Forecast, Webdienst, Online-Lernen.

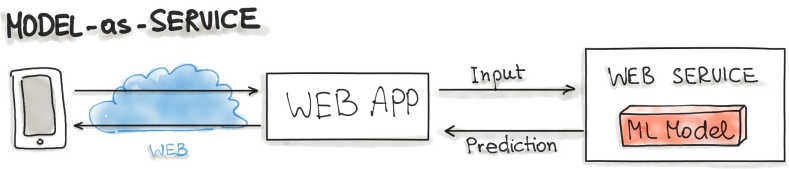

Abbildung 5.5: Model as Service Pattern (aus „Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow“, Kapitel 2 „End-to-End Machine Learning Project“, von Aurlien Gron)

Model as Dependency

Model as Dependency ist wahrscheinlich die einfachste Art, ein ML-Modell zu paketieren. Ein verpacktes ML-Modell wird als eine Dependency innerhalb der Software-Anwendung betrachtet und konsumiert, d.h., die Anwendung ruft die Vorhersagemethode auf und übergibt die Werte. Der Rückgabewert einer solchen Methodenausführung ist eine Vorhersage, die von dem zuvor trainierten ML-Modell durchgeführt wird. Der Model as Dependency-Ansatz wird meist für die Implementierung des Forecast-Musters verwendet.

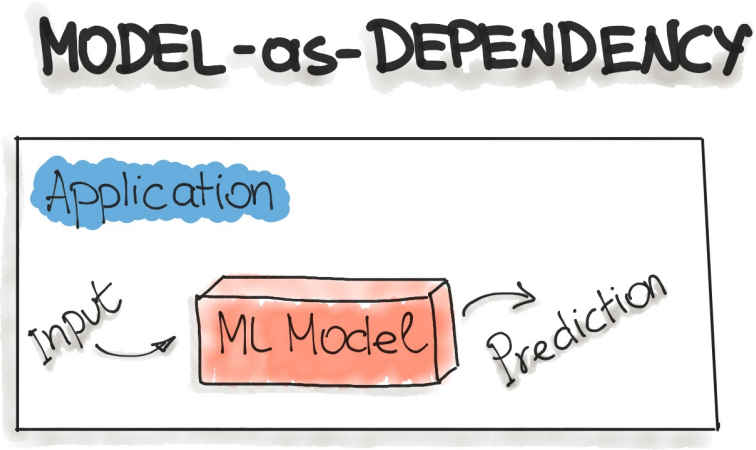

Abbildung 5.6: Model as Dependency

Precompute Serving

Diese Art des ML-Modell-Serving ist eng mit dem ML-Workflow „Forecast“ verbunden. Mit dem Serving-Muster „Precompute“ wird ein bereits trainiertes ML-Modell verwendet und die Vorhersagen für den eingehenden Daten vorberechnet. Die resultierenden Vorhersagen werden in der Datenbank persistiert. Daher wird

für jede Eingabeanforderung die Datenbank abgefragt, um das Vorhersageergebnis zu erhalten.

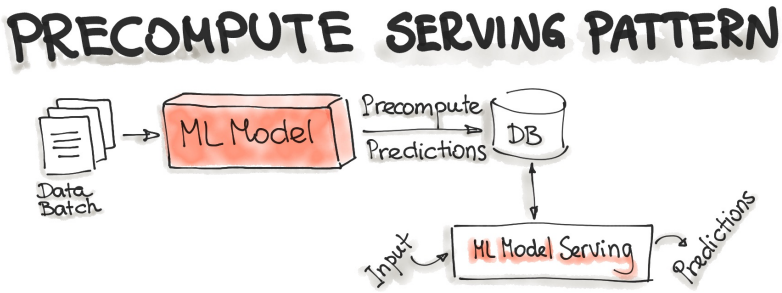

Abbildung 5.7: Precompute Serving Pattern

Zum Nachlesen: Bringing ML to Production[5]

Model on Demand

Das Model on Demand-Pattern behandelt das ML-Modell ebenfalls als eine Abhängigkeit, die zur Laufzeit verfügbar ist. Dieses ML-Modell hat im Gegensatz zum Model as Dependency-Muster einen eigenen Release-Zyklus und wird unabhängig veröffentlicht.

Für eine solche On-Demand-Modellbereitstellung wird typischerweise eine Message-Broker-Architektur verwendet. Das Architekturmuster der Message-Broker-Topologie enthält zwei Haupttypen von Architekturkomponenten: eine Broker-Komponente und eine Event-Prozessor-Komponente. Die Broker-Komponente ist der zentrale Teil, der die Event-Channels enthält, die innerhalb des Event-Flows verwendet werden. Die Event-Channels, die in der Broker-Komponente enthalten sind, sind Message-Queues. Wir können uns eine solche Architektur als Input- und Output-Queues vorstellen. Der Message-Broker

[5] https://www.slideshare.net/mikiobraun/bringing-ml-to-production-what-is-missing-amld-202o

erlaubt es einem Prozess, Vorhersageanfragen in eine Request-Queue zu schreiben. Der Event-Prozessor enthält die Modelllaufzeitumgebung und das Modell. Dieser Prozess verbindet sich mit dem Broker, liest die Anfragen als Batches aus der Request-Queue und sendet sie an das Modell, um die Vorhersagen zu erstellen. Der Model-Serving-Prozess führt die Vorhersageerstellung auf den Eingabedaten aus und schreibt die resultierenden Vorhersagen in die Prediction-Queue. Anschließend werden diese von dem Prozess konsumiert, der die Anfrage gestellt hat.

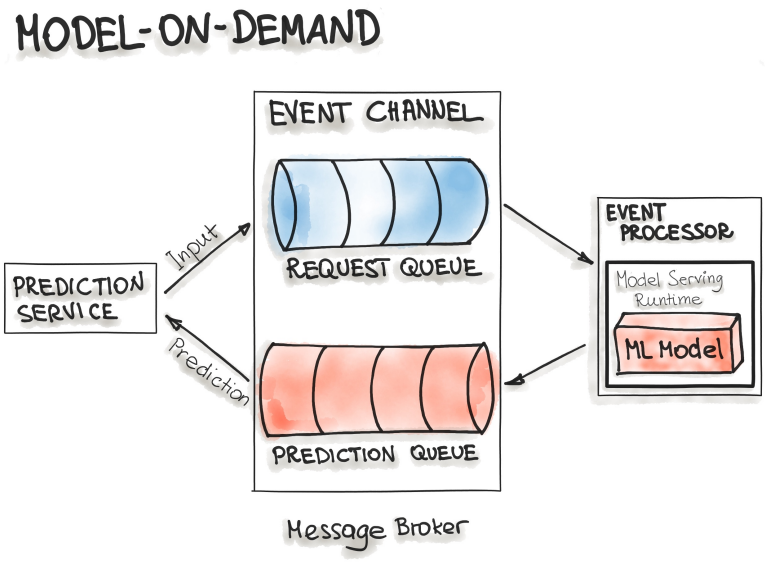

Abbildung 5.8: Model on Demand

Zum Nachlesen:

- Event-driven Architecture[6]

[6] https://learning.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

- Web Services vs. Streaming for Real-Time Machine Learning Endpoints[7]

Hybrid Serving (Federated Learning)

Federated Learning, auch bekannt als Hybrid Serving, ist eine weitere Möglichkeit, ein Modell bereitzustellen. Es ist einzigartig in der Art und Weise, dass es nicht nur ein Modell gibt, das das Ergebnis vorhersagt, sondern viele. Genauer gesagt, gibt es so viele Modelle, wie es Nutzende gibt, zusätzlich zu einem, das auf einem Server gehalten wird.

Das Modell auf der Serverseite wird nur einmal mit den Daten der realen Welt trainiert. Es legt das Ausgangsmodell für jeden Nutzenden fest. Außerdem ist es ein relativ allgemein trainiertes Modell, so dass es für die Mehrheit der Benutzer passt. Auf der anderen Seite gibt es die nutzungsseitigen Modelle. Aufgrund der steigenden Hardware-Standards bei mobilen Geräten ist es möglich, dass die Geräte ihre eigenen Modelle trainieren. So trainieren die Geräte ihr eigenes hochspezialisiertes Modell für ihren eigenen Nutzenden. In regelmäßigen Abständen senden die Geräte ihre bereits trainierten Modelldaten (nicht die persönlichen Daten) an den Server. Dort wird das Servermodell angepasst, so dass die aktuellen Trends der gesamten Nutzergemeinschaft durch das Modell abgedeckt werden.

Dieses Modell wird als neues Ausgangsmodell festgelegt, das von allen Geräten verwendet wird. Aus Nutzungsfreundlichkeit wird das Servermodell nur dann aktualisiert, wenn sich das Gerät im Leerlauf befindet, mit dem WiFi verbunden ist und geladen wird. Auch die Tests werden auf den Geräten durchgeführt, daher wird das neu übernommene Modell vom Server an die Geräte gesendet und auf Funktionalität getestet.

Der große Vorteil dabei ist, dass die für das Training und Testen verwendeten Daten, die sehr persönlich sind, die Geräte nie verlassen und trotzdem alle verfügbaren Daten erfasst werden. Auf diese Weise ist es möglich, hochpräzise Modelle zu trainieren, ohne große Mengen von (wahrscheinlich persönlichen) Daten in der Cloud speichern zu müssen.

[7] https://towardsdatascience.com/web-services-vs-streaming-for-real-time-machine-learning-endpoints-c08054e2b18e

Allerdings werden normale Machine-Learning-Algorithmen mit homogenen und großen Datensätzen auf leistungsfähiger Hardware gebaut, die immer für das Training zur Verfügung steht. Beim Federated Learning gibt es in dieser Hinsicht eventuell Probleme: Die mobilen Geräte sind weniger leistungsfähig, die Trainingsdaten sind über Millionen von Geräten verteilt und diese sind nicht immer für das Training verfügbar.

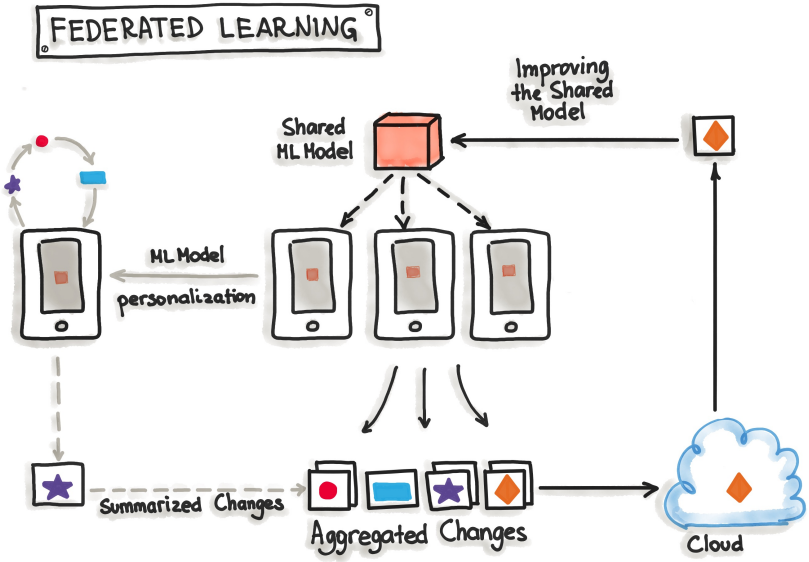

Abbildung 5.9: Federated Learning (aus „Federated Learning: Collaborative Machine Learning without Centralized Training Data“ von Brendan McMahan und Daniel Ramage)

5.3 Bereitstellungsstrategien

Bei den Gestaltungsmöglichkeiten der Bereitstellung hinsichtlich der Service-Infrastruktur beschränken wir uns auf zwei gängige Strategien, nämlich das Bereitstellen von ML-Modellen als Container und als Serverless-Funktionen.

Bereitstellen von ML-Modellen als Container

Bislang gibt es keine standardmäßige, offene Lösung für das ML-Modell-Deployment. Da die Inferenz von ML-Modellen als zustandslos, leichtgewichtig und idempotent angesehen wird, wird die Containerisierung zum De-facto-Standard für die Bereitstellung. Das bedeutet, dass wir einen Container bereitstellen, der ein ML-Modell umhüllt. Eine Möglichkeit besteht darin, den gesamten ML-Tech-Stack (samt Abhängigkeiten) und den Code für den Prediction-Service in einen Container zu packen. Die ML-Funktionalität, wie z. B. die Vorhersage, ist dann über z. B. eine REST API verfügbar.

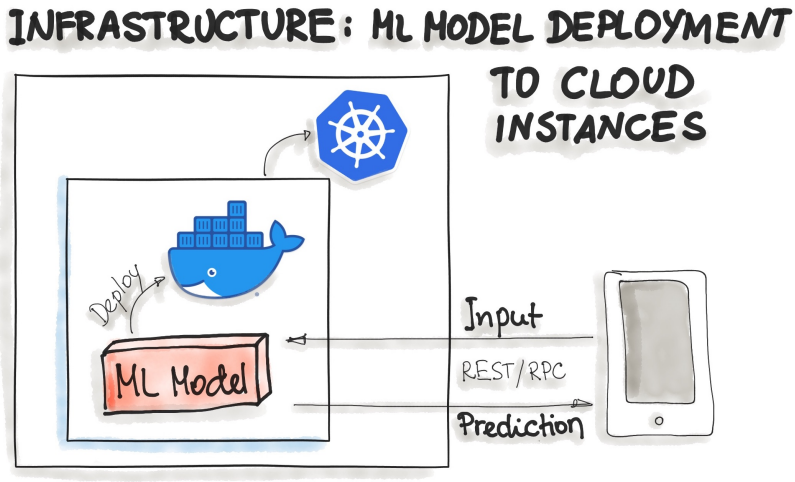

Abbildung 5.10: Container Infrastructure for Model Deployment

Bereitstellen von ML-Modellen als Serverless-Funktionen

Verschiedene Cloud-Anbieter bieten bereits Plattformen für ML an, über die ein Modell mit zugehörigen Diensten bereitgestellt werden kann. Beispiele sind AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio und

IBM Watson Machine Learning, um nur einige zu nennen. Weitere kommerzielle Cloud-Dienste bieten auch die Containerisierung von ML-Modellen an, wie AWS Lambda und Google App Engine Servlet Host.

Um ein ML-Modell als Serverless-Funktion bereitzustellen, werden Anwendungscode und Abhängigkeiten in ein Package verpackt, mit einer einzigen Einstiegspunkt-Funktion. Diese Funktion wird dann von dem Cloud-Anbieter entsprechend verwaltet (z. B. mit Azure Functions, AWS Lambda oder Google Cloud Functions). Es sollte jedoch auf mögliche Einschränkungen der bereitgestellten Artefakte geachtet werden, z. B. auf die Größe des Artefakts.

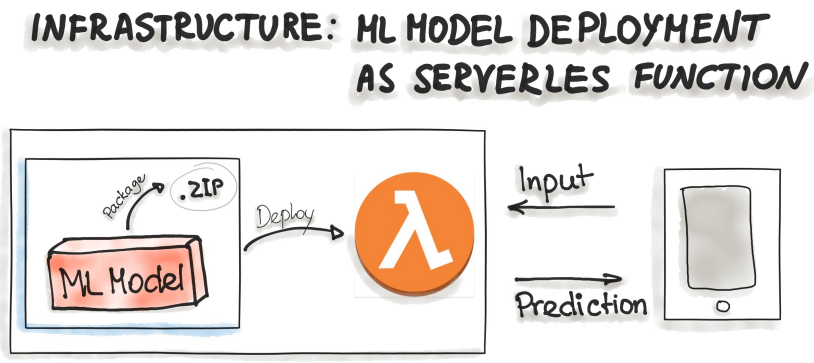

Abbildung 5.11: Serverless Infrastructure for Model Deployment

6 MLOps-Best Practices

6.1 Iterativ-inkrementelle Entwicklung

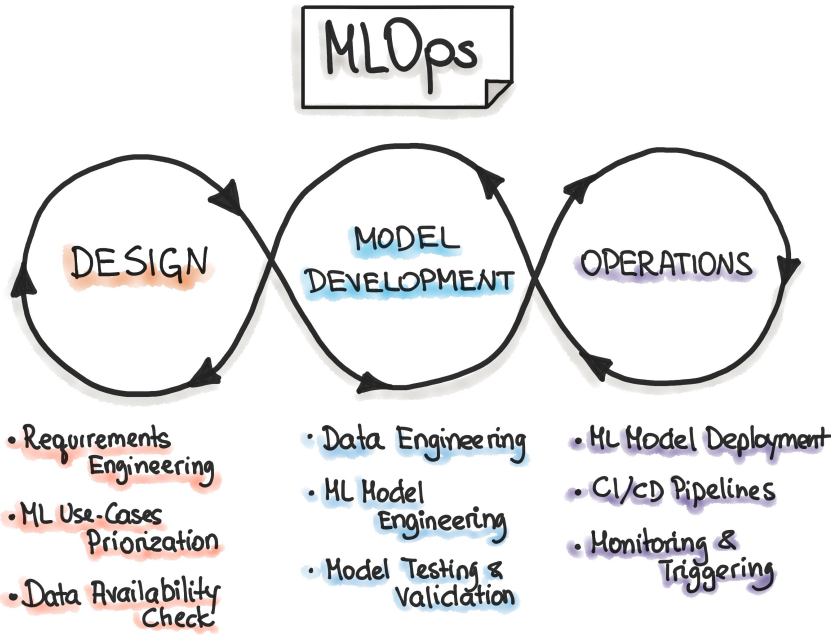

Abbildung 6.1: Agile ML-Workflow

Der komplette MLOps-Prozess umfasst drei Phasen: *Design der ML-gestützten Anwendung*, *ML-Experimentierung und Entwicklung* und *ML-Betrieb*.

Die erste Phase ist dem Geschäftsverständnis, dem Datenverständnis und dem Design der ML-gestützten Software gewidmet. In dieser Phase identifizieren wir unseren potenziellen Benutzer, entwerfen die ML-Lösung, um sein Problem zu lösen, und bewerten die weitere Entwicklung des Projekts. Meistens agieren wir innerhalb von zwei Problemkategorien - es geht entweder um Steigerung der

Produktivität des Anwenders oder um Erhöhung der Interaktivität unserer Anwendung.

Zu Beginn definieren wir ML-Anwendungsfälle und priorisieren sie. Die Best Practice für ML-Projekte ist es, an je einem Anwendungsfall der Reihe nachzuarbeiten. Die Entwurfsphase zielt darauf ab, die verfügbaren Daten zu inspizieren, die zum Trainieren unseres Modells benötigt werden, und die funktionalen und qualitativen Anforderungen an unser ML-Modell zu spezifizieren. Diese Anforderungen sollten wir nutzen, um die Architektur der ML-Anwendung zu entwerfen, die Bereitstellungsstrategie festzulegen und eine Test-Suite für das zukünftige ML-Modell zu erstellen.

Die nachfolgende Phase der ML-Experimentierung und -Entwicklung widmet sich der Verifizierung der Anwendbarkeit von ML für unser Problem durch die Implementierung eines Proof-of-Concept für das ML-Modell. Hier führen wir iterativ verschiedene Schritte durch, wie z. B. die Identifizierung oder die Anpassung des geeigneten ML-Algorithmus für unser Problem, das Data Engineering und das Model Engineering. Das primäre Ziel in dieser Phase ist es, ein ML-Modell von stabiler Qualität zu liefern, das wir in der Produktion einsetzen werden.

Das Hauptziel der Endphase, des ML-Betriebs, ist die Bereitstellung des zuvor entwickelten ML-Modells in der Produktionsumgebung unter Verwendung etablierter DevOps-Praktiken wie Testen, Versionierung, Continuous Deployment und Monitoring.

Alle drei Phasen sind miteinander verknüpft und beeinflussen sich gegenseitig. Zum Beispiel wird sich die Designentscheidung während der Designphase in die Experimentierphase fortpflanzen und schließlich die Bereitstellungsoptionen während der abschließenden Betriebsphase beeinflussen. Sie kann sich in einer dieser Phasen als falsch herausstellen und das Team zurück in die Designphase bringen. Diese iterative Vorgehensweise ist in ML-Projekten ganz natürlich und sollte mit Hilfe beschriebener MLOps Praktiken und einer automatisierten Pipeline robust, nachvollziehbar und effizient gestaltet werden.

6.2 Lose gekoppelte Architektur

Gene Kim et al. schreiben im Buch „Accelerate“, dass *„hohe Leistung [bei der Softwarebereitstellung] mit allen Arten von Systemen möglich ist, vorausgesetzt, dass die Systeme - und die Teams, die sie erstellen und warten - lose gekoppelt sind. Diese wichtige architektonische Eigenschaft ermöglicht es den Teams, einzelne Komponenten oder Dienste problemlos zu testen und bereitzustellen, selbst wenn die Organisation und die Anzahl der von ihr betriebenen Systeme wachsen - das heißt, sie ermöglicht es Organisationen, ihre Produktivität zu steigern, während sie skalieren.“*

Zusätzlich empfehlen Gene Kim et al. *„eine lose gekoppelte Architektur zu verwenden. Dies betrifft das Ausmaß, in dem ein Team seine Anwendungen bei Bedarf testen und bereitstellen kann, ohne dass eine Orchestrierung mit anderen Diensten erforderlich ist. Eine lose gekoppelte Architektur ermöglicht es Ihren Teams, unabhängig zu arbeiten, ohne auf andere Teams für Support und Services angewiesen zu sein, was es ihnen wiederum ermöglicht, schnell zu arbeiten und Mehrwert für das Unternehmen zu liefern.“*

Bei ML-basierten Softwaresystemen kann es schwieriger sein, eine lose Kopplung zwischen Komponenten zu erreichen als bei traditionellen Softwarekomponenten. ML-Systeme haben in mehrfacher Hinsicht schwache Komponentengrenzen. Zum Beispiel können die Ausgaben von ML-Modellen als Eingaben für ein anderes ML-Modell verwendet werden, und solche verschachtelten Abhängigkeiten können sich beim Training und Testen gegenseitig beeinflussen.

Eine grundlegende Modularität kann durch die Strukturierung des ML-Projekts erreicht werden. Um eine Standard-Projektstruktur zu erstellen, empfehlen wir die Verwendung von speziellen Vorlagen, wie z. B. die in der folgenden Liste:

- Cookiecutter datenwissenschaftliche Projektvorlage[1]
- Die Vorlage für den Data Science Lifecycle-Prozess[2]
- PyScaffold[3]

[1] https://drivendata.github.io/cookiecutter-data-science/
[2] https://github.com/dslp/dslp-repo-template
[3] https://github.com/pyscaffold/pyscaffold

6.3 Automatisierung

Der Grad der Automatisierung der Daten-, ML-Modell- und Code-Pipelines bestimmt den Reifegrad des ML-Prozesses. Mit zunehmender Reife wird auch die Geschwindigkeit für das Training neuer Modelle erhöht. Das Ziel eines MLOps-Teams ist es, die Bereitstellung von ML-Modellen im Kernsoftwaresystem oder als Servicekomponente vollständig zu automatisieren, d.h. alle ML-Workflow-Schritte ohne manuelle Eingriffe durchführen zu können.

Die Automatisierung kann graduell in drei Stufen eingeführt werden, angefangen mit manuellem Modelltraining und Deployment und abschließend mit der automatischen Ausführung von ML- und CI/CD-Pipelines (siehe auch „MLOps: Continuous delivery and automation pipelines in machine learning“[4]).

1. *Manueller Prozess.* Dies ist ein typischer Data-Science-Prozess, der zu Beginn der Implementierung von ML durchgeführt wird. Diese Stufe hat einen experimentellen und iterativen Charakter. Jeder Schritt in jeder Pipeline, wie z. B. Datenvorbereitung und -validierung, Modelltraining und -test, wird manuell ausgeführt. Die übliche Vorgehensweise ist die Verwendung von Rapid Application Development (RAD) Tools, wie z. B. Jupyter-Notebooks.
2. *ML-Pipeline-Automatisierung.* Die nächste Stufe führt die automatische Ausführung des Modelltrainings und das kontinuierliche Training des Modells ein. Immer wenn neue Daten verfügbar sind, wird der Prozess des Modelltrainings ausgelöst. Diese Automatisierungsebene umfasst auch die Schritte der Daten- und Modellvalidierung.
3. *CI/CD-Pipeline-Automatisierung.* In der letzten Stufe führen wir ein CI/CD-System ein, um schnelle und zuverlässige ML-Modellbereitstellungen in der Produktion durchzuführen. Der Hauptunterschied zum vorherigen Schritt besteht darin, dass wir nun automatisch die Daten, das ML-Modell und die ML-Trainingspipeline-Komponenten erstellen, testen und bereitstellen.

Das folgende Bild zeigt die automatisierte ML-Pipeline mit CI/CD-Routinen:

Die MLOps-Schritte, die den Prozess der Automatisierung der ML-Pipeline widerspiegeln, werden in der folgenden Tabelle erläutert:

[4] https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#top_of_page

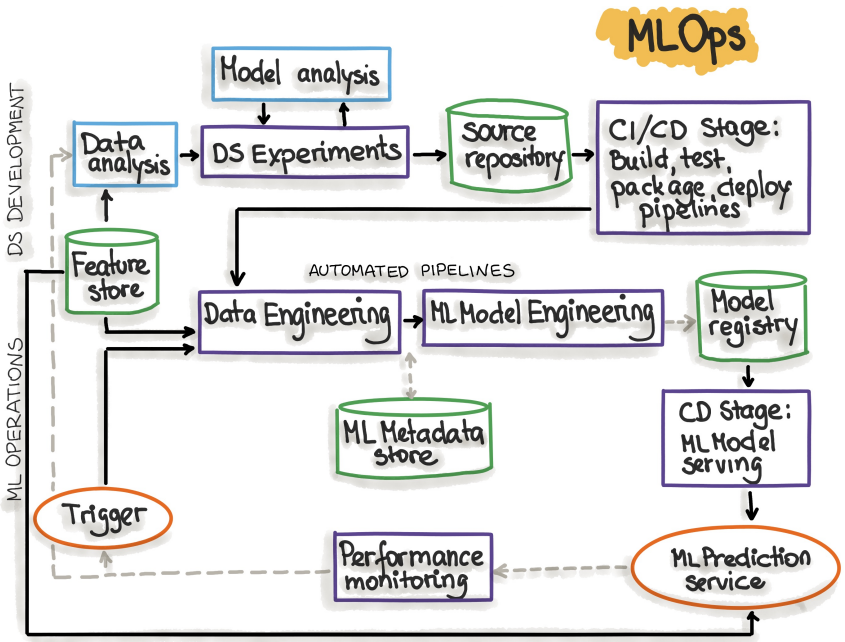

Abbildung 6.2: Automatisierte ML Pipeline (aus „MLOps: Continuous delivery and automation pipelines in machine learning“)

MLOps-Schritt	Ergebnis
Entwicklung & Experimentierung (ML-Algorithmen, neue ML-Modelle)	Sourcecode für die Pipelines für Extrahieren, Validieren und Vorbereiten der Daten, Modelltraining, Modellauswertung, Modelltests
Pipeline Continuous Integration (Kompilieren des Sourcecode und Ausführen der Tests)	Pipeline Komponenten für die Bereitstellung der Pakete und Executables
Pipeline Continuous Delivery (Bereitstellung-Pipelines für die Zielumgebung)	Neuimplementierung des Modells im Betrieb
Automatisierte Trigger (Pipeline wird in Produktion automatisch ausgeführt, es werden Zeitpläne und Trigger eingesetzt)	Trainiertes Modell wird im Modell-Registry gespeichert
Model Continuous Delivery (Modell-Serving für die Vorhersage)	Der mit dem Modell vorhersagende Service ist im Betrieb (z.B. Modell als REST API bereitgestellt)
Monitoring (Datensammeln über und Überwachung der Modellleistung auf Live-Daten)	Trigger für den Start der Pipelines oder den Neustart der Pipelines für die nächste Iteration der Experimente

Abbildung 6.3: MLOps-Schritte

Für das MLOps-Setup werden mehrere Komponenten installiert oder vorbereitet. In der folgenden Tabelle sind diese Komponenten aufgeführt:

MLOps-Setup-Komponenten	Beschreibung
Source Control	Versionierung von Code, Daten und ML-Modellartifakten
Test & Build Services	CI Tools für (1) Qualitätssicherung aller ML-Artefakte und (2) Modellpaketierung, d.h. Bereitstellung der Pakete und Executables für weitere Pipelines
Deployment Services	CD tools für Pipelines, die in die Zielumgebung deployen
Modell-Registry	Ein Register für trainierte ML-Modelle
Feature Store	Vorverarbeitung der Daten als Features, die dann an Modelltraining-Pipelines und Modell-Serving weitergegeben werden
ML Metadata Store	Speicher für Metadaten aus dem Modelltraining, z.B Modellname, Parameter, Trainingsdaten, Testdaten, Ergebnisse der Metriken
ML Pipeline Orchestrator	Automatisierung der Schritte der ML-Experimentierung.

Abbildung 6.4: MLOps-Komponenten

6.4 Versionierung

Das Ziel der Versionierung ist es, Nachvollziehbarkeit und Reproduzierbarkeit im ML-Workflow sicherzustellen. Jede ML-Modellspezifikation sollte in einem VCS verwaltet werden und eine eigene Versionsnummer erhalten. Diese wird hochgezählt, sobald ein Neutraining des Modells stattfindet. Auslöser für ein Modell-Retraining können Kalenderereignisse, Messaging, Überwachungsereignisse sowie Änderungen an Daten, Modelltrainingscode und Anwendungscode sein. Die häufigsten Gründe für ein Neutraining des Modells sind laut SIG MLOps[5]) die folgenden:

- Modelle können auf der Grundlage neuer Trainingsdaten neu trainiert werden.
- Modelle können auf der Grundlage neuer Trainingsansätze neu trainiert werden.
- Modelle können selbstlernend sein.
- Modelle können sich mit der Zeit verschlechtern.
- Modelle können in neuen Anwendungen eingesetzt werden.
- Modelle können Angriffen ausgesetzt sein und müssen überarbeitet werden.

Versionierung der Modelle bringt weiterhin Vorteile in folgenden Situationen:

[5] https://lists.cd.foundation/g/sig-mlops

- Modelle können schnell auf eine frühere Version zurückgesetzt werden.
- Die Einhaltung von Unternehmens- oder behördlichen Vorschriften kann eine Prüfung oder Untersuchung sowohl des ML-Modells als auch der Daten erfordern, daher benötigen wir Zugriff auf alle Versionen des produzierten ML-Modells.
- Daten können sich auf mehreren Systemen befinden.
- Daten dürfen juristisch nur in bestimmten Rechtsräumen gespeichert werden.
- Datenspeicher ist nicht immutable und es können ungewollte Änderungen passieren.
- Dateneigentum kann ein Faktor sein.

Zum Nachlesen: How do we manage ML Models? Modell-Management-Frameworks[6]

6.5 Nachvollziehbarkeit der Experimente

Entwicklung im ML-Bereich ist ein hochgradig iterativer und forschungszentrierter Prozess. Im Gegensatz zum traditionellen Softwareentwicklungsprozess können in der ML-Entwicklung mehrere Modelle parallel trainiert werden, bevor die Entscheidung getroffen wird, welches Modell in die Produktion überführt wird.

Die Experimente während der ML-Entwicklung sollten strukturiert durchgeführt werden. Eine Möglichkeit, mehrere Experimente zu verfolgen, besteht darin, verschiedene Branches im VCS zu verwenden, die jeweils einem separaten Experiment gewidmet sind. Die Ausgabe eines jeden Branches ist ein trainiertes Modell. Abhängig von der gewählten Metrik werden die trainierten ML-Modelle miteinander verglichen und das passende Modell wird ausgewählt.

Eine solche reibungsarme Verzweigung wird z. B. vollständig vom Tool DVC[7] unterstützt, das eine Erweiterung von Git und ein Open-Source-Versionskontrollsystem für Machine-Learning-Projekte ist. Ein weiteres beliebtes Tool zur Verfolgung von ML-Experimenten ist die Bibliothek Weights

[6] https://www.inovex.de/blog/machine-learning-model-management/
[7] https://dvc.org/

and Biases (wandb)[8], die automatisch die Hyperparameter und Metriken der Experimente verfolgt.

6.6 Testing

Automatisiertes Testen hilft, Probleme schnell und in einem frühen Stadium zu entdecken. Das ermöglicht eine schnelle Behebung von Fehlern und das Lernen aus Fehlern.

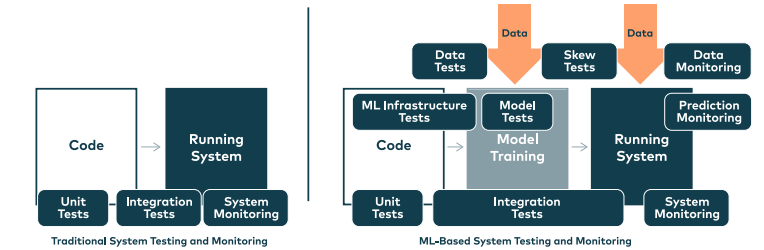

Abbildung 6.5: Testen in ML-Systemen, aus „The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction" von E.Breck et al. 2017

Wir unterscheiden drei Bereiche für das Testen in ML-Systemen: *Tests für Modell-Features und -Daten, Tests für die Modellentwicklung und Tests für die ML-Infrastruktur*. Folgende Checklisten sollen helfen, diese Bereiche vollständig mit automatisierten Tests abzudecken.

6.6.1 Tests für Modell-Features und -Daten (Checkliste)

- Daten-Validierung: automatische Prüfung der Daten und des Features-Schemas.

[8] https://www.wandb.com/

    - Um ein Schema zu erstellen, berechnen Sie Statistiken aus den Trainingsdaten. Dieses Schema kann als *Expectation Definition* or *Semantic Role* für Eingabedaten während der Trainings- und Bereitstellungsphasen verwendet werden.

- Test der Wichtigkeit von Features, um zu verstehen, ob neue Features die Vorhersagekraft erhöhen.

    - Berechnen Sie den Korrelationskoeffizienten für die Spalten der Features.
    - Trainieren Sie das Modell mit einem oder zwei Features.
    - Trainieren Sie einen Satz von verschiedenen Modellen, indem Sie jeweils eine Teilmenge der Features benutzen, die jeweils ein Feature auslässt.
    - Messen Sie Datenabhängigkeiten, Inferenzlatenz und RAM-Nutzung für jedes neue Feature. Vergleichen Sie sie mit der Vorhersagekraft der neu hinzugefügten Features.
    - Entfernen Sie ungenutzte/veraltete Features aus Ihrer Infrastruktur und dokumentieren Sie dies.

- Funktionen und Datenpipelines sollten richtlinienkonform sein (z. B. GDPR). Diese Anforderungen sollten sowohl in Entwicklungs- als auch in Produktionsumgebungen programmatisch überprüft werden.

- Der Code zur Erstellung von Modell-Features sollte durch Unit-Tests getestet werden (um Fehler in Features zu erfassen).

6.6.2 Tests für eine zuverlässige Modellentwicklung (Checkliste)

Wir müssen spezifische Testunterstützung für die Erkennung von ML-spezifischen Fehlern bereitstellen.

- Tests für die Übereinstimmung mit den Geschäftszielen.

    - Das Testen im Model Training sollte Routinen beinhalten, die überprüfen, ob die von den Algorithmen getroffenen Entscheidungen mit den Geschäftszielen übereinstimmen. Das bedeutet, dass die Verlustmetriken des ML-

Algorithmus (MSE, log-loss, etc.) mit den Metriken für die Geschäftsauswirkungen (Umsatz, Benutzerbindung etc.) korrelieren sollten.
    - Die Beziehung zwischen Verlustmetriken und Auswirkungsmetriken kann in kleinen A/B-Tests mit einem absichtlich verschlechterten Modell gemessen werden.
    - *Zum Nachlesen:* Auswahl der richtigen Metrik zur Bewertung von Machine-Learning-Modellen, Teil 1[9], Teil 2[10]
- Modellveralterungstest.
    - Das Modell wird als veraltet definiert, wenn das trainierte Modell keine aktuellen Daten enthält und/oder die Anforderungen an die Geschäftsauswirkungen nicht erfüllt. Veraltete Modelle können die Qualität der Vorhersage in intelligenter Software beeinträchtigen.
    - Führen Sie A/B-Experimente mit älteren Modellen durch. Durch das Beobachten der Vorhersagequalität mit zunehmendem Alter der Modelle kann man ein Verständnis dafür bekommen, wie oft das ML-Modell neu trainiert werden sollte.
- Abschätzung der Kosten für anspruchsvollere ML-Modelle.
    - Die Leistung eines ML-Modells sollte mit einem einfachen ML-Basismodell verglichen werden (z. B. lineares Modell vs. neuronales Netzwerk).
- Validieren der Leistung eines Modells.
    - Die Autoren von „Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology“[11] empfehlen, die Teams und Verfahren, die die Trainings- und Testdaten sammeln, zu trennen, um die Abhängigkeiten zu beseitigen und zu vermeiden, dass sich falsche Methodik vom Trainingsdatensatz auf den Testdatensatz überträgt.
    - Verwenden Sie einen zusätzlichen Testdatensatz, der mit den Trainings- und Validierungssätzen disjunkt ist. Verwenden Sie diesen Testdatensatz nur für die abschließende Bewertung.

[9] https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4

[10] https://medium.com/usf-msds/choosing-the-right-metric-for-evaluating-machine-learning-models-part-2-86d5649a5428

[11] https://arxiv.org/pdf/2003.05155.pdf

- Fairness-/Bias-/Inklusionstest für die ML-Modellleistung.
  - Sammeln Sie mehr Daten, die potenziell unterrepräsentierte Kategorien enthalten.
  - Untersuchen Sie die Eingabe-Features dahingehend, ob sie mit geschützten Benutzerkategorien korrelieren.
  - *Zum Nachlesen:* „Tour of Data Sampling Methods for Imbalanced Classification“[12].
- Konventionelle Unit-Tests für jede Feature-Erstellung, Modellspezifikationscode und Tests.
- Tests für Model Governance.

6.6.3 ML-Infrastruktur-Test (Checkliste)

- Das Training der ML-Modelle sollte reproduzierbar sein, d.h., das Training des ML-Modells auf denselben Daten sollte identische ML-Modelle erzeugen.
  - Diff-Testing von ML-Modellen setzt ein deterministisches Training voraus, das aufgrund der Nichtkonvexität der ML-Algorithmen, der zufälligen Seed-Generierung oder des verteilten ML-Modelltrainings schwer zu erreichen ist.
  - Maßnahme: Bestimmen Sie die nichtdeterministischen Teile in der Codebasis für das Modelltraining und versuchen Sie, den Nichtdeterminismus zu minimieren.
- Testen Sie die ML-API, auch mit Stresstests.
  - Unit-Tests zur zufälligen Generierung von Eingabedaten und zum Training des Modells für einen einzelnen Optimierungsschritt (z. B. Gradientenabstieg).
  - Crash-Tests für das Modelltraining. Das ML-Modell sollte nach einem Absturz mitten im Training von einem Prüfpunkt aus wiederhergestellt werden.
- Testen Sie die algorithmische Korrektheit.

[12] https://machinelearningmastery.com/data-sampling-methods-for-imbalanced-classification/

    - Unit-Test, der sicherstellt, dass das ML-Modell nicht zu Ende trainiert werden soll, sondern die Verlustfunktion schon nach einigen Iterationen und während des gesamten Trainings abnimmt.
    - Vermeiden Sie Diff-Tests mit zuvor erstellten ML-Modellen, da solche Tests schwer zu pflegen sind.
- Integrationstests: Die gesamte ML-Pipeline sollte Integrationstests unterzogen werden.
    - Erstellen Sie einen vollautomatischen Test, der regelmäßig die gesamte ML-Pipeline anstößt. Der Test sollte validieren, dass die Daten und der Code jede Stufe des Trainings erfolgreich abschließen und das resultierende ML-Modell die erwartete Leistung erbringt.
    - Alle Integrationstests sollten ausgeführt werden, bevor das ML-Modell die Produktionsumgebung erreicht.
- Validieren des ML-Modells, bevor es bereitgestellt wird.
    - Legen Sie einen Schwellenwert fest und testen Sie die Modellqualität auf langsame Verschlechterung über viele Versionen auf einem Validierungsset.
    - Legen Sie einen Schwellenwert fest und testen Sie auf einen plötzlichen Leistungsabfall in einer neuen Version des ML-Modells.
- ML-Modelle werden vor der Auslieferung überprüft.
    - Testen Sie, dass ein ML-Modell erfolgreich in den Produktionsbetrieb geladen wird und die Vorhersage auf realen Daten wie erwartet generiert wird, bevor Sie die frühere Version des Modells abschalten.
- Testen, ob das Modell in der Trainingsumgebung das gleiche Ergebnis liefert wie das bereitgestellte Modell.
    - Der Unterschied zwischen der Leistung auf den „Holdout“-Daten und den „Nextday“-Daten ist zu beobachten. Ein gewisser Unterschied wird immer bestehen. Achten Sie auf große Leistungsunterschiede zwischen „holdout“- und „nextday“-Daten, da dies ein Hinweis darauf sein kann, dass einige zeitabhängige Features eine Verschlechterung des ML-Modells verursachen.
    - Stellen Sie sicher, dass das Modell für dieselben Eingabedaten dasselbe Ergebnis in der Trainingsumgebung und in der produktiven Umgebung vorhersagt.

6.7 Monitoring

Sobald das ML-Modell bereitgestellt wurde, muss es überwacht werden, um sicherzustellen, dass das ML-Modell die erwartete Leistung erbringt. Die folgende Checkliste für Modellmonitoring in der Produktion wurde aus „The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction“ von E. Breck et al. 2017[13] übernommen:

- Überwachung von Dependencies in der gesamten Pipeline führen zu einer Benachrichtigung, z. B. bei Änderung der Datenversion, Änderungen im Code oder beim Aktualisieren anderer Abhängigkeiten.
- Überwachen Sie Dateninvarianten in Dateneingaben: es bedarf einer Warnung, wenn Daten nicht mit dem Schema übereinstimmen, das im Trainingsschritt festgelegt wurde.
- Überwachen Sie, ob Trainings- und Serving-Features denselben Wert berechnen.
    - Da die Generierung von Trainings- und Serving-Features an physisch getrennten Orten stattfinden kann, muss sorgfältig geprüft werden, ob diese unterschiedlichen Codepfade logisch identisch sind.
    - Vorgehen: Protokollieren Sie eine Stichprobe des Serving-Verkehrs. Berechnen Sie Verteilungsstatistiken (min, max, avg, % der fehlenden Werte usw.) für die Trainingsmerkmale und die Serving-Merkmale und stellen Sie sicher, dass sie übereinstimmen.
- Überwachen Sie die numerische Stabilität des ML-Modells, z. B. sollte beim Auftreten von NaNs oder Unendlichkeiten ein Alarm ausgelöst werden.
- Überwachen Sie die Rechenleistung des ML-Systems. Sowohl dramatische als auch schleichende Rückschritte in der Rechenleistung sollten gemeldet werden.
    - Sammeln Sie Metriken zur Systemnutzung wie GPU-Speicherzuweisung, Netzwerkverkehr und Festplattennutzung. Diese Metriken sind nützlich für die Abschätzung der Cloud-Kosten.
- Überwachen Sie die Veralterung des Systems in Produktion.

[13] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf

    - Messen Sie das Alter des Modells.
    - Die Modellüberwachung ist ein kontinuierlicher Prozess, daher ist es wichtig, die Elemente für die Überwachung zu identifizieren und eine Strategie für die Modellüberwachung zu erstellen, bevor das Modell in die Produktion geht.
- Überwachen Sie die Prozesse der Feature-Generierung, da sie Auswirkungen auf das Modell haben. Wiederholen Sie die Feature-Generierung in regelmäßigen Abständen.
- Überwachen Sie die Verschlechterung der Vorhersagequalität des ML-Modells. Sowohl dramatische als auch schleichende Verschlechterungen der Vorhersagequalität sollten gemeldet werden. Eine Verschlechterung kann z. B. aufgrund von Änderungen in den Daten auftreten.
    - Messen Sie die statistische Verzerrung (Bias) in den Vorhersagen. Die Modelle sollten nahezu keine Verzerrungen aufweisen.
    - Wenn ein Label unmittelbar nach der Vorhersage verfügbar ist, können wir die Qualität der Vorhersage in Echtzeit messen und Probleme identifizieren.

Die folgende Abbildung zeigt, dass die Modellüberwachung implementiert werden kann, indem die Präzision, der Recall und der F1-Score der Modellvorhersage zusammen über die Zeit verfolgt werden. Die Abnahme der Präzision, des Recalls und des F1-Scores löst ein erneutes Training des Modells aus, was zu einer Wiederherstellung der Leistung des Modells führt.

6.8 Reproduzierbarkeit

Reproduzierbarkeit in einem ML-Workflow bedeutet, dass jede Phase der Datenverarbeitung, des Modelltrainings und des Modelleinsatzes bei gleichem Input identische Ergebnisse liefern sollte. Die folgende Tabelle gibt eine Übersicht über mögliche Probleme und Lösungen hinsichtlich der Reproduzierbarkeit.

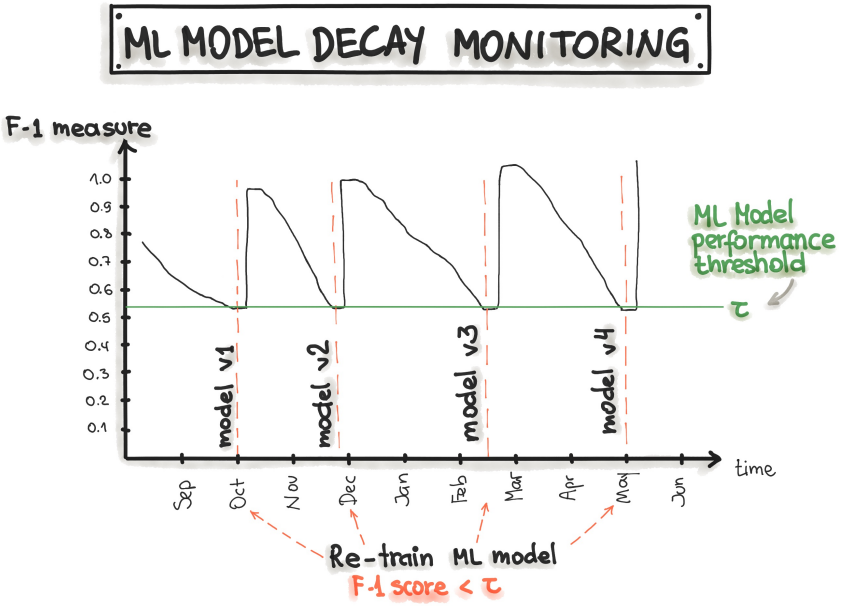

Abbildung 6.6: Monitoring des Modellverfalls

Phase	Problem	Sicherstellung der Reproduzierbarkeit
Sammeln der Daten	Generierung der Trainingsdaten kann nicht reproduziert werden (z.B. weil die Datenbank ständig verändert wird oder Daten zufällig geladen werden)	1) Machen Sie immer einen Backup der Daten. 2) Speichern Sie einen Snapshot der Daten, z.B. in der Cloud. 3) Datenquellen sollten mit Zeitstempel versehen werden, sodass ein vollständiges Bild der Daten zu jeglichem Zeitpunkt einsichtbar ist 4) Versionieren Sie die Daten
Feature Engineering	Szenarien: 1) Fehlende Daten werden mit zufälligen Werten oder Mittelwerten gefüllt 2) Labels werden aufgrund von einem Teil der Beobachtungen entfernt 3) Nichtdeterministische Feature-Extraktion	1) Feature-Generierung-Code muss versioniert werden 2) Stellen Sie die Reproduzierbarkeit im Punkt "Sammeln der Daten" sicher
Modelltraining / -paketierung	Nichtdeterminismus	1) Stellen Sie sicher, dass Features immer dieselbe Reihenfolge haben. 2) Dokumentieren und automatisieren Sie die Transformationen der Features, z.B. die Normalisierung. 3) Dokumentieren und automatisieren Sie die Wahl der Hyperparameter. 4) Im Ensemble Learning: Dokumentieren und automatisieren Sie die Kombination der ML-Modelle
Modell-Deployment	1) Für das Training wurde eine andere Version der Software benutzt als im Produktionssystem. 2) Für das Modell notwendige Daten stehen in der produktiven Umgebung nicht zur Verfügung	1) Software Versionen und Abhängigkeiten müssen mit der produktiven Umgebung übereinstimmen. 2) Benutzen Sie eine Container-Technologie und spezifizieren Sie die Container. 3) Im Idealfall sollte die Programmiersprache im Training und Deployment dieselbe sein

Abbildung 6.7: Reproduzierbarkeit

6.9 Einschätzung der Produktionsreife

Der „ML Test Score“ misst die Gesamtproduktionsreife eines ML-Systems (siehe „The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction“ by E.Breck et al. 2017[14]). Dieser wird wie folgt errechnet:

- Für jeden Test wird ein halber Punkt vergeben, wenn der Test manuell ausgeführt wird und die Ergebnisse dokumentiert und verteilt werden.
- Ein voller Punkt wird vergeben, wenn ein System vorhanden ist, das den Test automatisch und wiederholt ausführt.
- Addieren Sie die Punktzahl für jeweils vier Abschnitte einzeln: Datentests, Modelltests, ML-Infrastrukturtests und Monitoring.
- Das Minimum der für jeden der Abschnitte aggregierten Punktzahlen ist der endgültige ML Test Score.

Nach der Berechnung des ML Test Scores können wir eine Aussage über die Produktionsreife des ML-Systems treffen. Die folgende Tabelle gibt die Interpretation der möglichen Punktzahl wieder:

Punktzahl	Beschreibung
0	Eher ein Forschungsprojekt als ein Produktionssystem.
(0,1]	Ein wenig getestet, allerdings sind große Lücken und Risiken zu erwarten, was die Zuverlässigkeit des Systems angeht.
(1,2]	Grundlegendes Mindestniveau für ein Produktionssystem, aber es sollte mehr in Testing, Monitoring und Automatisierung im Allgemeinen investiert werden.
(2,3]	Ganz gut getestet, aber wahrscheinlich gibt es Potenzial im Bereich der Automatisierung.
(3,5]	Automatisches Testen und Monitoring auf hohem Niveau.
>5	Automatisches Testen und Monitoring auf vorbildlichem Niveau.

Abbildung 6.8: ML Test Score

[14] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/aad9f93b86b7addfea4c419b9100c6cdd26cacea.pdf

6.10 ML Software Delivery Metriken

In der jüngsten Studie über den Stand von DevOps[15] haben die Autoren vier Schlüsselmetriken hervorgehoben, die die Effektivität der Softwareentwicklung und -bereitstellung von Hochleistungsunternehmen erfassen: Deployment Frequency, Lead Time for Changes, Mean Time To Restore und Change Fail Percentage. Diese Metriken haben sich als nützlich erwiesen,[16] um die eigene ML-basierte Softwarebereitstellung zu messen und zu verbessern. In der folgenden Tabelle geben wir die Definition jeder der Metriken an und stellen die Verbindung zu MLOps her.

Metrik	DevOps	MLOps
Deployment Frequency	Wie oft wird in Ihrem Unternehmen Code für die Produktion bereitgestellt oder für Endbenutzer freigegeben?	Die Frequenz der ML-Modellbereitstellung hängt ab von 1) den Anforderungen an das Modelltraining (von weniger häufig bis hin zum Online Learning). Zwei Aspekte sind für das Neutrainieren des Modells entscheidend: 1.1) Metrik für Modellverfall 1.2) Verfügbarkeit von neuen Daten 2) Der Grad der Automatisierung des Bereitstellungsprozesses, der zwischen "manueller Bereitstellung" und "voll automatisierter CI/CD-Pipeline" liegen kann
Lead Time for Changes	Wie lange dauert es von dem Moment der Speicherung des Codes im Versionverwaltungssystem bis zum erfolgreichen Einsatz des Codes in der Produktion?	Lead Time for Changes für ML-Modelle hängt ab von 1) der Dauer der explorativen Phase in der Datenanalyse, um das ML-Modell für den Einsatz/die Bereitstellung fertig zu stellen. 2) der Dauer des Modelltrainings 3) der Anzahl und der Dauer der manuellen Schritte im Bereitstellungsprozess
Mean Time To Restore (MTTR)	Wie lange dauert es im Allgemeinen, bis der Dienst wiederhergestellt ist, wenn eine Störung oder ein Defekt auftritt, der sich auf die Nutzer auswirkt (z.B. ein ungeplanter Ausfall oder eine Beeinträchtigung des Dienstes)?	MTTR für ML-Modelle hängt ab von der Anzahl und der Dauer der manuellen Schritte im Modell-Debugging und in der Modellbereitstellung. Falls das Modell neu trainiert werden muss, hängt MTTR auch ab von der Dauer des Trainings. Alternativ dazu bezieht sich MTTR auf die Dauer des Rollbacks des ML-Modells auf die vorherige Version
Change Failure Rate	Wie viel Prozent der Änderungen an der Produktion oder der Releases für die Benutzer führen zu einer Beeinträchtigung des Dienstes (z.B. zu einer Beeinträchtigung oder einem Ausfall des Dienstes) und erfordern anschließend eine Korrektur (z.B. einen Hotfix, Rollback, Fix Forward, Patch)?	Change Failure Rate für ML-Modelle kann durch die Differenz zwischen den Leistungskennzahlen des aktuell eingesetzten ML-Modells und denen des vorherigen Modells ausgedrückt werden, z.B. Precision, Recall, F-1, accuracy, AUC, ROC, False Positives usw. Die Fehlerquote bei ML-Modelländerungen steht auch im Zusammenhang mit A/B-Tests

Abbildung 6.9: Software-Delivery-Metriken

Um die Effektivität des ML-Entwicklungs- und -Deployment-Prozesses zu verbessern, sollte man die oben genannten vier Schlüsselmetriken messen. Ein praktischer Weg, höhere Effektivität zu erreichen, besteht darin, zuerst die CI/CD-Pipeline zu implementieren und testgetriebene Entwicklung für die Daten-, ML-Modell- und Softwarecode-Pipelines einzuführen.

[15] https://services.google.com/fh/files/misc/state-of-devops-2019.pdf
[16] https://www.thoughtworks.com/radar/techniques/four-key-metrics

7 Zusammenfassung

Die komplette ML-Entwicklungspipeline umfasst drei Ebenen, auf denen während des Lebenszyklus Änderungen auftreten können: **Daten, ML-Modell** und **Code.** Das bedeutet, dass bei Systemen, die auf maschinellem Lernen basieren, der Auslöser für einen Build die Kombination aus einer Codeänderung, einer Datenänderung oder einer Modelländerung sein kann. Die folgenden Tabellen fassen die MLOps Best Practices zusammen. Diese helfen, die technischen Schulden Ihres ML-Projekts unter Kontrolle zu halten und eine zuverlässige Leistung des ML-Modells im Sinne der Geschäftsziele sicherzustellen.

MLOps-Bereich	Daten	ML-Modell	Code
Versionierung	1) Pipelines für die Vorbereitung von Daten 2) Features Store 3) Datensätze 4) Metadaten	1) Pipeline für ML-Modelltraining 2) ML-Modell (selbst) 3) Hyperparameter 4) Experimente	1) Applikationscode 2) Konfigurationen
Tests	1) Datenvalidierung (Finden von Fehlern in Daten) 2) Unit-Tests für Erzeugung der Features	1) Unit-Tests für Modellspezifikation 2) Integrationstests für ML-Modelltraining-Pipeline 3) Validierung des ML-Modells vor der Operationalisierung 4) ML-Modellverfalltests in Produktion 5) Tests für ML-Modellrelevanz und Korrektheit 6) Tests für Qualitätsanforderungen (Sicherheit, Fairness, Interpretabilität)	1) Unit-Tests 2) Integrationtests für die End to End-Pipeline
Automatisierung	1) Datentransformationen 2) Feature-Erzeugung und -Manipulation	1) Daten-Engineering-Pipeline 2) ML-Modelltraining-Pipeline 3) Hyperparameter-/Parameterauswahl	1) ML-Modelldeployment mit CI/CD 2) Herstellung der Anwendung

Abbildung 7.1: Zusammenfassung der Best Practices

MLOps-Bereich	Daten	ML-Modell	Code
Reproduzierbarkeit	1) Backup 2) Datenversionierung 3) Extrahierung von Metadaten 4) Versionierung des Prozesses des Feature Engineering	1) Hyperparameterwahl ist identisch in dev und prod 2) Reihenfolge der Features is gleich 3) Ensemble Learning: Kombination der ML-Modelle ist gleich 4)Pseudocode des Modells ist dokumentiert	1) Versionen aller Abhängigkeiten in dev and prod sind identisch 2) Technischer Stack für dev and prod Umgebungen ist gleich 3) Ergebnisse sind durch Bereitstellen der Container-Abbilder oder virtueller Maschinen reproduzierbar
Deployment	1) Feature Store für dev and prod Umgebungen	1) Containerisierung des ML-Stack 2) REST API 3) On-premise, Cloud oder Edge	1) On-premise, Cloud oder Edge
Monitoring	1) Änderungen in der Datenverteilung (Training- vs. Serving-Daten) 2) Training vs. Serving Features	1) ML-Modellverfall 2) Numerische Stabilität 3) Leistung des ML-Modells	1) Qualität der Vorhersagen der Anwendung auf realen Daten
Dokumentation	1) Datenquellen 2) Entscheidungen, z. B. wie und woher die Daten kommen sollen 3) Labelling-Methoden	1) Modellauswahlkriterien 2) Design der Experimente 3) Pseudocode des Modells	1) Deployment-Prozess 2) Wie der Code lokal auszuführen ist
Projekt-/Code-Struktur	1) Datenordner für rohe und verarbeitete Daten 2) Ordner für Daten-Engineering-Pipeline 3) Ordner für Tests der Daten-Engineering-Methoden	1) Ordner mit dem trainierten Modell 2) Ordner für Notebooks 3) Ordner für Feature Engineering 4) Ordner für ML-Modell-Engineering	1) Ordner für Bash/Shell-Skripte 2) Ordner für Tests 3) Ordner für Deployment-Files (z. B. Dockerfiles)

Abbildung 7.2: Zusammenfassung der Best Practices

8 Unser Angebot

Der allgegenwärtige Hype um Machine Learning und Artificial Intelligence kann den Eindruck vermitteln, dass Software mit ML-Modellen einfach zu entwickeln ist. Laut diversen Studien scheitern allerdings fast 80 % der ML-Projekte.

Wir unterstützen Sie gerne bei Ihrem Machine-Learning-Entwicklungsvorhaben – von der frühen Erkundungsphase, in der Produktentwicklung, bis hin zu allen Betriebsthemen.

Training: Domain-driven Design für ML-Produkte

Sie stehen vor der Aufgabe, innovative, datengetriebene Softwareprodukte zu entwickeln und haben bereits erste praktische Erfahrung mit Machine Learning gemacht? Dann stellt sich jetzt die Frage: An welcher/n Problemstelle/n kann in der Produktentwicklung Machine Learning gewinnbringend eingesetzt werden?

Lernen Sie von unseren Trainer*innen praxisorientiert, wie Sie AI/ML Use Cases finden und verifizieren können und erhalten Sie von ihnen die richtigen Tools und Vorgehensweisen bis zur Produktfinalisierung. Dabei bedienen wir uns effektiver Domain-driven-Design-Methodik (keine Vorkenntnisse notwendig).

Mehr Infos und Termine: https://www.socreatory.com/de/trainings/ddd4ml

Wir beraten ehrlich, denken innovativ und entwickeln leidenschaftlich gern. Das Ergebnis: Erfolgreiche Softwarelösungen, Infrastrukturen und Geschäftsmodelle.

Als Technologieunternehmen fokussieren wir uns auf Strategie- und Technologieberatung, Softwarearchitektur und -entwicklung, Methoden- und Technologietraining sowie Plattform-Infrastrukturen.

Wir unterstützen mit über 170 Mitarbeiter*innen an Standorten in Deutschland und der Schweiz Unternehmen und Organisationen bei Konzeption und Umsetzung komplexer Vorhaben und der Verbesserung bestehender Softwaresysteme.

Wir engagieren uns in Open-Source-Projekten sowie dem iSAQB e.V., und geben Wissen und Erfahrungen auf Konferenzen und Meetups sowie in zahlreichen Büchern und Fachartikeln weiter.

Über die Autorinnen

Nadejda Ismailova

Nadejda ist Consultant bei INNOQ. In ihrem Informatikstudium legte sie den Fokus auf den Bereich Machine Learning. Aktuell beschäftigt sie sich mit einer Vielzahl an Themen der Softwareentwicklung, unter anderem mit Security-Themen wie Authentifizierung und Autorisierung (OAuth2, OIDC).

Anja Kammer

Anja ist Senior Consultant bei INNOQ und entwickelt cloud-native Web-Anwendungen. Sie beschäftigt sich mit Automatisierung von Deployments und CI/CD Systemen im Speziellen. Ihre Schwerpunkte liegen in den Bereichen DevOps, Cloud Infrastruktur und Kubernetes.

@kammer_anja

Dr. Larysa Visengeriyeva

Larysa hat im Bereich Augmented Data Quality an der TU Berlin promoviert. Bei INNOQ beschäftigt sie sich mit der Operationalisierung von Machine Learning (MLOps), Datenarchitekturen wie Data Mesh und Domain-Driven Design. Sie ist außerdem Maintainerin von ml-ops.org

@visenger

Das Training von ML-Modellen kann viel Zeit in Anspruch nehmen. Die eigentliche Herausforderung aber ist die Integration eines ML-Systems in die Produktionsumgebung, das heißt in das Softwareprodukt, mit dem die Nutzer:innen interagieren.

Ein ML-System besteht dabei aus drei Hauptelementen: den Trainingsdaten, dem ML-Modell und dem Code für das Training der Modelle. Wir nutzen die DevOps-Prinzipien für ML-Systeme (MLOps), um die ML-Entwicklung (Development) und den ML-Betrieb (Operations) zu kombinieren. Als Erweiterung von DevOps widmet sich MLOps der Automatisierung und Überwachung in allen Schritten der Integration von ML-Systemen in Softwareprojekten.

In diesem Primer erklären wir die Grundlagen und Prinzipien von MLOps mit dem Ziel, die MLOps-Prozesse aus der Engineering-Perspektive zu vermitteln.

innoq.com

INNOQ