FUJITSU

Whitepaper
Model Management mit MLflow

Management und Data Governance sind zentrale Themen von Digital-Strategien. Nur so kann sichergestellt werden, dass alle Abteilungen auf gleichen, aktuellen, qualitäts- und quellengesicherten Ständen arbeiten. Dennoch scheitern derartige Ansätze in der Praxis häufig. Dieser Beitrag zeigt die relevanten Aspekte einer Weiterverwendung der Daten für Business Intelligence- und Data Science-Projekte auf und legt exemplarisch mit Hilfe des Open Source Tools MLflow dar, wie Unternehmen auch in diesen Bereichen die erforderliche Governance und das notwendige Life Cycle Management erreichen können.

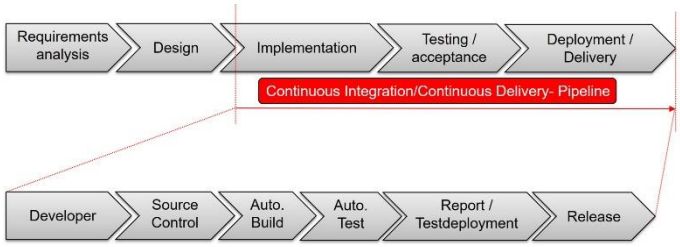
Inhalt

Einleitung

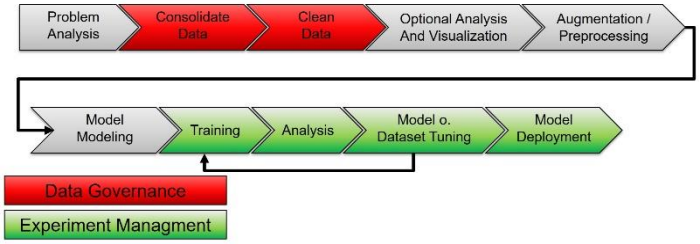
Viele Firmen haben inzwischen verstanden, dass Data Management und Data Governance zentrale Themen einer jeden Digital-Strategie sein müssen. Denn nur so können sie sicherstellen, dass alle Abteilungen auf gleichen, aktuellen, qualitäts- und quellengesicherten Ständen arbeiten. Doch trotz dieser richtigen Grundüberlegung scheitern derartige Ansätze in der Praxis häufig, weil sie zu kurz gedacht sind – auch die Verwendung muss gemanagt werden. Der Beitrag zeigt die relevanten Aspekte einer Weiterverwendung der Daten für Business Intelligence- und Data Science-Projekte auf und legt exemplarisch mit Hilfe des Open Source Tools MLflow dar, wie Unternehmen auch in diesen Bereichen die erforderliche Governance und das notwendige Life Cycle Management erreichen können.

Nach dem Aufsetzen entsprechender Prozesse für Data Management und Data Governance folgt in der Praxis nicht selten die Ernüchterung. Denn dann passiert folgendes: Der Data Scientist – oft frisch von der Uni – lädt alles, was er für sein Projekt braucht, auf den eigenen Rechner, da es sich so einfacher und schneller arbeiten lässt. Anschließend kämpft er sich durch themenbezogenes Data Cleaning und ETL (Extract, Transform, Load) und probiert sein Register an Algorithmen aus. Ist er erfolgreich und schafft es, eine relevante Frage für einen Fachbereich zu beantworten, bekommt er danach in regelmäßigen Intervallen die Daten des Fachbereichs per E-Mail, lässt sein Skript neu laufen und mailt das Ergebnis zurück. Am Ende entstehen dadurch die gleichen Probleme, die auch bei einem Datenmanagement ohne konsequente Governance-Strategie auftreten: Die Quelle des Modells ist unklar, die Qualität und Aktualität der zugrundeliegenden Daten ist fragwürdig und Änderungen in der Methodik zwischen den Auswertungen sind nicht nachvollziehbar.

Model Management ist relevant für jedes Unternehmen, das aus Daten abgeleitete Modelle produktiv einsetzen will. Der Wert der Daten ergibt sich erst durch ihre Weiterverwendung. Wenn diese nicht den gleichen Ansprüchen genügt wie die Datensammlung, ist das Gesamtergebnis leider meist immer noch mit den gleichen Problemen behaftet, die ursprünglich zur Einführung einer Data Governance-Strategie geführt haben. Mit der steigenden Wichtigkeit von Machine Learning und Künstlicher Intelligenz (KI) in allen Bereichen eines Unternehmens, sollte sich also jeder CTO (oder auch Chief Digital Officer oder Chief Data Officer) auch mit Strategien und Prozessen befassen, um diese Nachvollziehbarkeit der Modellentstehung sicherzustellen.



In klassischen Softwareprojekten gibt es eine sogenannte Continuous Integration/Continuous Delivery Pipeline, die sicherstellt, dass neu geschriebener Code kontinuierlich automatisiert produktiviert werden kann.



Eine solche Prozess-Pipeline, die sicherstellt, dass eine kontinuierliche Weiterentwicklung von Machine Learning (ML)-Modellen gewährleistet ist und zu einer schnellen Produktivierung führt, sollte ebenfalls Standard in jedem ML-Projekt werden.

Wie funktioniert ein Data Science-Experiment?

Data Science trägt nicht nur zufällig den Namen Science. Es geht um wissenschaftliche Experimente auf Grundlage von gesammelten Daten. Dabei gilt: Experimente müssen dokumentiert und wiederholbar sein. Nur so kann man sich auf die resultierenden Modelle verlassen, diese überprüfen und bei Problemen auf Ursachenforschung gehen. Spätestens dann, wenn ein Modell produktiv eingesetzt werden soll, um Entscheidungen zu unterstützen oder gar autonom Eingriffe vorzunehmen, muss seine Herkunft vollständig nachvollziehbar und möglicherweise sogar auditierbar sein.

Der Datensatz als Grundlage

Ein Data Science Experiment startet immer mit der Auswahl der Daten für das Modelltraining. Da alle Machine Learning und KI-Methoden letztendlich auf die eine oder andere Art statistische Auswertungen der zugrundeliegenden Trainingsdaten sind, stellen diese einen entscheidenden Teil des Experiments dar und müssen akribisch dokumentiert werden. Die „gleichen" Daten von einem anderen Zeitraum oder aus einer anderen Quelle können stark unterschiedliche Modelle ergeben.
Idealerweise werden die Daten in einem entsprechenden Data-Managementsystem verwaltet und diese Meta-Informationen sind dort bereits hinterlegt. Falls nicht, sollten sie zumindest für das Projekt mit dokumentiert werden. Auch der Zeitpunkt, zu dem der Datensatz für das Modelltraining aus dem zentralen System gezogen wurde, muss festgehalten werden, um auch im Falle von Daten-Updates im Hauptsystem die korrekte Datengrundlage rekonstruieren zu können.

Use Case-bezogenes ETL

Stammen die Daten aus einem System, welches ein sauberes Data Management betreibt, so sind sie vermutlich bereits vorprozessiert und qualitätsgesichert und diese Schritte sind im Rahmen des Managementprozesses auch dokumentiert. Trotzdem bedarf fast jeder Use Case auch einer eigenen Datenvorverarbeitung. Das können simple Aspekte sein, wie etwa das Beschränken der Daten auf einen bestimmten Typ oder Bereich, aber auch komplexere Schritte, wie das Aufteilen in Kategorien oder das Ersetzen von fehlenden Werten, zum Beispiel durch lokale Mittelwerte.

Alle Vorverarbeitungsschritte festzuhalten ist nicht nur für die Nachvollziehbarkeit wichtig, sondern auch, damit Data Engineers später bei der Inferenz die gleichen Schritte auf die Daten anwenden und so unverfälschte Ergebnisse erhalten können. Da es hier auf die exakten Veränderungen ankommt, ist es nötig, den verwendeten Code

vorzuhalten. Ein gutes Governance-System schließt daher neben dem Code des Modelltrainings auch den der Preprocessing-Schritte mit ein.

Das eigentliche Modelltraining

Hat der Data Scientist seine Daten zufriedenstellend vorbereitet, kann er nun ein Modell mit diesen trainieren. Dieses Modelltraining besteht meistens aus dem Trainieren und Vergleichen mehrerer unterschiedlicher Modelle und Parametereinstellungen - etwa mittels eines Grit-Search aller Kombinationen.
Auch hier empfiehlt es sich grundsätzlich, den exakten ausgeführten Code des Experiments abzuspeichern. Gleichzeitig ist es für eine bessere Übersicht und Wartbarkeit sinnvoll, das letztendlich ausgewählte Modell mit seinen Tuning-Parametern, sowie verschiedener Performance-KPIs auf einem gesonderten Validierungsset, zusätzlich zu dokumentieren. So können der Data Scientist und ein späterer Betrieb effizient kontrollieren, ob es bei einer erneuten Trainingsausführung (beispielsweise aufgrund aktuellerer Daten oder mehr Rechenleistung) zu Performanceveränderungen kommt oder sich eine andere Art von Modell plötzlich als geeigneter herausstellt.

Inferenz

Ist ein Modell fertig trainiert und hat in der Validierung eine hinreichend gute Performance gezeigt, so soll es natürlich angewandt werden, um auf neuen Daten basierend Vorhersagen zu treffen. Dieser Schritt nennt sich Inferenz. Hier ist es wichtig, später zurückführen zu können, welches Modell für eine Vorhersage verwendet wurde. Bei seltenen oder sehr wichtigen Vorhersagen dokumentiert man also am besten bei jeder Modell-Anwendung das Modell, z.B. über eine Versionsnummer. Bei Massenanwendungen, wie zum Beispiel Empfehlungen in einem Online-Shop, hingegen reicht es normalerweise zu dokumentieren, in welchem Zeitraum ein Modell jeweils für eine bestimmte Anwendung produktiv geschaltet war.

Produktivierung

Inferenz findet oftmals nicht nur in den Analystics-Abteilungen der Unternehmen statt, sondern auch in Softwarekomponenten in Industrie, e-Commerce und Medizin. Hier reicht es nicht aus, sich lediglich damit zu befassen, wie ein Modell in die Software beziehungsweise auf das jeweilige Device kommt. Es gilt vielmehr, für alle Komponenten eine kontinuierliche Entwicklungspipeline zu schaffen, dementsprechend auch für trainierte Modelle.

Zusammenfassung Experimentbeschreibung

Eine gute Experimentbeschreibung besteht aus dem zugrundeliegenden Datensatz (mit Entnahmedatum), dem Code der Vorverarbeitungsschritte, dem Code des Modelltrainings und optional beschreibenden Daten (Parameter, Performance) zu dem erzeugten Modell. Zusätzlich ist mindestens zu dokumentieren, wann das Modell wo produktiv im Einsatz war.

Model Management Tools

Aus der Motivation heraus, standardisierte Prozesse und Arbeitsumgebungen zu schaffen, haben sich in der Data Science-Gemeinde diverse Tools und Ansätze entwickelt, welche die Arbeit eines Data Scientists einfacher machen. Exemplarisch dafür seien hier drei für den Bereich Model Management Umfeld genannt: MLFlow, Sacred und DVC (Data Science Version Control).

DVC

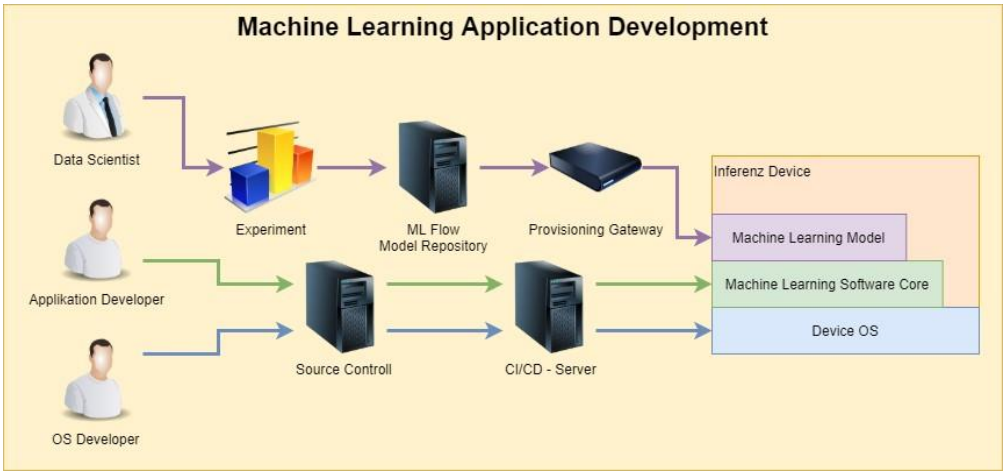
Data Science Version Control ist stark an Source Code-Versionierungstools wie zum Beispiel GIT angelehnt. Wenn der Data Scientist sein Experiment verändert, also Hyperparameter verändert, ein anderes Modell benutzt oder ein anderes Preprocessing implementiert, geht der Ansatz von DVC davon aus, dass der veränderte Source auch versioniert wird. DVC sorgt dabei dafür, dass auch die Ergebnisse des Experiments entsprechend der Source-Versionierung abgelegt werden. DVC hat hierbei nicht den Anspruch auch als Analysetool fungieren zu wollen, sondern beschränkt sich auf einfache Kommandos.

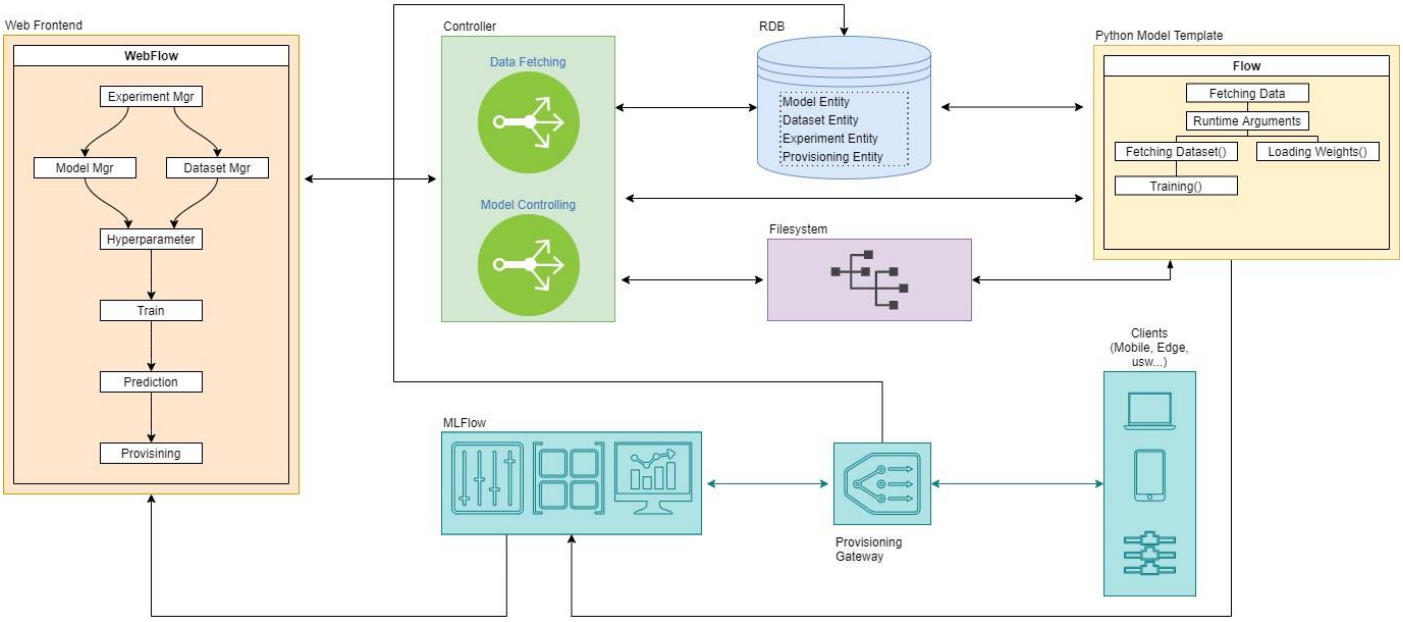
Sacred

Sacred ist ein in Python geschriebenes Tool, das dem Data Scientist neben dem Experiment-Tracking eine aufwendige Nutzeroberfläche mit vielen Analysemöglichkeiten zur Verfügung stellt. Die zu historisierenden Daten werden im Experiment annotiert und entsprechend bei der Ausführung eines Experiments in eine NoSQL-Datenbank geschrieben. Anschließend können die Experimente über die Sacred-eigene Benutzeroberfläche verglichen und analysiert werden.

MLFlow

MLFlow ermöglicht neben der Historisierung von Experimenten, deren Parameter und Metriken auch deren Vergleich. Es besteht aus einem Node.JS Core, einem Web-Frontend, einer API und Libraries für diverse Programmiersprachen wie Python, R oder Java. Diese Libraries





ermöglichen das Tracking der Daten. MLFlow kann lokal für Tests oder als Server-Variante im Unternehmenskontext genutzt werden. Diverse Analysemöglichkeiten über die MLFlow-eigene Benutzeroberfläche sowie die Möglichkeit, Experimente komplett zu steuern runden den Programmumfang ab.

Die schon erwähnte API (Application Programming Interface) ist allerdings einer der entscheidenden Vorteile, denn über diese ist MLFlow von außen steuerbar. Zusätzlich können über sie Artefakte wie etwa die trainierten Modelle angefordert werden. MLFlow bietet somit Unterstützung im gesamten Lebenszyklus eines Modells.

	DVC	Sacred	MLFlow
Entwicklungs-sprachen Support	Universal	Python	Native: Python, R, Java Rest-API: Universal
Tracking	Shell Scripte	Annotation	Libraries
Storage	File Storage	MongoDB+ File Storage	PostgresDB, SQLLite, S3(AWS, MinIO)
User Interface	Command Line	Web-Oberfläche	Web-Oberfläche
API	Nein	Nein	Ja
Model Storage	Ja	Ja	Ja
License	Apache 2	MIT	Apache 2

Im weiteren Verlauf dieses Beitrags wird das Vorgehen am Beispiel MLFlow beschrieben.

Die Umsetzung mit MLflow

Wie lassen sich diese Aspekte nun komfortabel verwalten, ohne zu viel zusätzliche Arbeit zu verursachen? Eine generische Software-Architektur, die sowohl für den Data Scientist als auch für den Konsumenten der erzeugten Daten Mehrwerte bietet, basiert auf zwei Säulen: dem Model Management und einem flexiblen User Interface.

Im Folgenden wird ein Architekturvorschlag mit geringer Komplexität, aber großem Mehrwert skizziert.

Meta-Daten-Tracking

Das Tracking an sich ist denkbar einfach. Als erstes legt der Data Scientist in seinem Trainingsskript den „Uniform Resource Identifier" (URI) fest unter dem der MLflow-Server erreichbar ist. Dies ermöglicht die Kommunikation mit dem Server. Dann verortet er sein Experiment, indem er eine Experimentbezeichnung vergibt. Experimente mit der gleichen Bezeichnung werden strukturell in MLflow zusammengelegt, wodurch sie verglichen und Änderungen nachverfolgt werden können.

Das Tracking kann individuell oder spezialisiert auf verschiedene Machine Learning Stacks erfolgen. So gibt es unter anderem für Pytorch, Tensorflow, Keras und SciKit-Learn Standart-Tracker, die alle wichtigen Daten automatisch aufzeichnen. Es gibt jedoch auch die Möglichkeit, selbst festzulegen, was erfasst werden soll sowie Methoden zur Aufzeichnung von Parametern, Artefakten und Metriken. Hat der Data Scientist ein Experiment abgeschlossen, kann er dieses mittels eines modernen webbasierten Frontends analysieren und mit älteren Versuchen vergleichen.

Die Architekturkomponenten im Detail

Die Architektur sieht vier Komponenten vor:

- Das Frontend/User Interface, mit dem Experimente gesteuert werden können
- Den Controller, der Befehle vom Frontend entgegennimmt und dafür sorgt, dass Skripte ausgeführt werden und nötige Daten aus der Datenbank beschafft werden
- Eine Standard-Datenbank, die Metadaten zu den Experimenten enthält, auf die hier nicht weiter eingegangen wird
- Das Backend, mit den Skripten für Training und Inferenz und dem MLflow-Wrapper für die Provisionierung

Das Interface

Analysen für die Optimierung der Experimente finden direkt im User Interface von MLflow statt. Die vorgeschlagene Architektur soll jedoch nicht nur dem Data Scientist gerecht werden, sondern auch den

Fachabteilungen, die etwa eine in Auftrag gegebene Regression später auf einfache Weise selbst nutzen wollen. Daher ist ein flexibles, rollenbasiertes Frontend erforderlich, welches den Workflow vor der Zusammenstellung des Experiments bis hin zur Inferenz abbildet.

Grundsätzlich erfordert ein solches Frontend individuelle Anpassungen, mit – je nach Rolle – unterschiedlichen Detailgraden. Der Data Scientist muss alle Experimente überblicken, Ergebnisse vergleichen und will vielleicht auch das Training aus dem Frontend steuern können. Die Fachabteilung braucht in der Regel nur Zugriff auf die Inferenz.

Der Controller

Die Abfragen und Eingaben aus dem Frontend werden an den Controller gesendet, der das Zusammenspiel der Komponenten regelt. Neben der Kontrolle über die Ausführung der Modelle, verbindet die Komponente Frontend, Datenbank und das Backend miteinander, hat also ebenso Gateway-Funktionalität. Software-technisch ist ein solcher Controller im einfachsten Falle ein Rest-Interface, welches Parameter entgegennimmt, in einem definierten Format an ein Ziel weiterleitet und etwas Logik beinhaltet. Wird – wie im Data Science Umfeld üblich – mit virtuellen Umgebungen wie Conda oder auch Container-basiert gearbeitet, muss vor der Weiterleitung das Ziel noch initialisiert werden. Sprich die virtuelle Umgebung muss geladen werden und erst dann kann das Zielskript ausgeführt werden. Der Controller ist ebenso verantwortlich für den Rückkanal der Ergebnisse zum Frontend.

Das Backend

Modelle können in unterschiedlichen Sprachen und Laufzeitumgebungen vorliegen. Um diese Skripte sinnvoll in eine Architektur zu integrieren, müssen sie von außen steuerbar gemacht werden. Das heißt, dass bestimmte Funktionen innerhalb des Skripts von außen aktiviert werden können.
Hierzu gibt es zwei gängige Ansätze. Die entsprechenden Funktionen können über einen API-Endpoint „as a Service" zur Verfügung gestellt werden, dann sind die Funktionen global von jedem nutzbar, der die Berechtigung, den Zugriff und die Schnittstellenbeschreibung hat.
Eine zweite, einfachere Methode, ist, das Skript mit Argumenten zu starten: das sind Parameter, die beim Aufruf des Programms übergeben werden. Hier braucht das System, welches die Funktionalität starten soll, Zugriff auf das Betriebssystem des entsprechenden Rechners oder der Virtuellen Maschine (VM). Um die Systemanforderung gering zu halten ist die Argument-Methode der einfachere Weg.

Ob das Modell gerade trainiert wird oder ob ein trainiertes Modell eine Vorhersage oder Klassifizierung durchführen soll, muss von außen steuerbar sein. Das Skript muss also strukturell in zwei Bereiche unterteilt werden, die über Argumente ansteuerbar sind.

Für das Training ist eine Reihe von Parametern notwendig: Es muss festgelegt werden ob das Modell schon vorinitialisiert werden soll, es müssen sämtliche Hyperparameter und Quellen der Trainings-, Validierungs- und Testdaten übergeben werden, sowie eine Experimentkennung für das Tracking der Daten. Diese Daten können auch im Vorfeld in einer Datenbank abgelegt und so stattdessen lediglich die ID des Datensatzes als Parameter übergeben werden.

Beim Trainieren müssen die Daten geladen und für das Modell transformiert werden. Anschließend wird das Modell erzeugt und das Training mit dem Ziel gestartet, vorgegebene KPIs wie Loss oder Accuracy zu optimieren. Bei zufriedenstellenden Ergebnissen wird das trainierte Modell gespeichert.

Für eine Inferenz sieht der Aufruf denkbar simpel aus: Dem Skript muss nur mitgeteilt werden, wie das Modell initialisiert wird und wo die Daten für die Inferenz zu finden sind, dann wird das Modell auf die Daten angewandt und das Ergebnis zurückgeliefert.

Das Skript-Template

Ein Modell-Skript, dass in der vorgeschlagenen Architektur verfügbar gemacht werden soll, muss folgende Funktionalitäten besitzen:

- Entsprechend der Startparameter entweder Trainings- oder Inferenzfunktion aufrufen
- In der Trainingsfunktion
  - die Daten aus der übergebenen Datenquelle laden
  - das Modell erzeugen
  - ggf. Gewichte für das Modell laden
  - Verbindung zu MLflow aufbauen
  - zu einem Experiment zuordnen
  - MLflow Tracking ausführen
  - Training starten
- In der Inferenz
  - Modell aufbauen
  - Gewichte des trainierten Modells laden
  - Vorhersage für die übergebene Datenquelle treffen
  - Ergebnis zurückliefern

Der Arbeitsalltag

Der Data Scientist kann das Skript in seiner bevorzugten Programmiersprache und IDE (Integrated Development Environment) entwickeln. Dann wird das Skript mit allen nötigen Metadaten, wie etwa Ort, Startargumente et cetera, in der Datenbank registriert. Nun steht das Modell im Frontend zur Verfügung und Trainings können über das User Interface gestartet werden. Während des Trainings werden Parameter, Artefakte und Metriken über MLflow historisiert. Ist das Training beendet, wird das Frontend aktualisiert, so dass der Data Scientist die Qualität überprüfen kann. Danach kann das Modell für Inferenz genutzt werden oder zur Produktivsetzung an andere Systeme verteilt werden.

Die zu implementierenden Komponenten sind so unspezifisch, dass sie in jeder Entwicklungssprache und jeder Infrastruktur umgesetzt werden können. Das heißt, es ist problemlos möglich, diese Architektur „Cloud native" oder „on premises" umzusetzen. Cloud Native-Implementierungen von MLflow werden bereits von Microsoft vorangetrieben und auch für AWS (Amazon Web Services) gibt es verschiedene Ansätze, MLflow effizient einzusetzen. Die Standardisierung der Modellentwicklung, der Analysetools, der Provisionierung und der Möglichkeit Machine Learning Funktionalität auch Leihen verfügbar zu machen, sind für den praktischen Einsatz in Unternehmen unschätzbare Vorteile.

Weiterführende Links:

- MLflow-Website: https://mlflow.org/
- GitHub-Projekt zu MLflow: https://github.com/mlflow/mlflow
- Informationen zur Implementierung von MLflow auf Microsoft Azure: https://docs.microsoft.com/de-de/azure/machine-learning/how-to-use-mlflow

Die Autoren:

Dr. Lisa Wagner arbeitet als Senior Systems Architect bei Fujitsu im Bereich Connected Services Manufacturing and Automotive. Zuvor war sie unteranderem als Principal Data Scientist bei einem auf Data Science spezialisiertem Start-Up tätig. Seit mittlerweile über fünf Jahren beschäftigt sie sich in Kundenprojekten insbesondere mit dem produktiven Einsatz von Machine Learning- und Deep Learning - Methoden.

Marcel Naujeck ist Senior Solution Achitect bei Fujitsu im Bereich Connected Services Manufacturing and Automotive. Er verfügt über langjährige Erfahrungen als Systemanalytiker und Enterprise Architect in verschiedenen Geschäftsfeldern. Als Innovation Engineer fokussierte sich seine Tätigkeit intensiv auf den Bereich Machine Learning sowie andere Methoden der künstlichen Intelligenz.

Kontakt
FUJITSU
Fujitsu Technology Solutions GmbH
Mies-van-der-Rohe-Strasse 8, 80807 München, Deutschland
Telefon: +49 1805 372-900*
E-Mail: cic@ts.fujitsu.com
Website: http://de.fujitsu.com
2012-04-01 CEMEA&I DE
* (Pro Anruf 14 Cent/Min.; die Preise für Anrufe aus dem Mobilfunknetz sind auf 42 Cent/Min. festgelegt worden)