

Projektdokumentation: Brustkrebs-Klassifikation

Ziel des Projektes:

Das Ziel dieses Projekts ist es, praktische Erfahrungen in der Entwicklung einer vollständigen MLOps-Pipeline für ein reales maschinelles Lernproblem – die Klassifikation von Brustkrebsdaten – zu sammeln. Im Mittelpunkt steht dabei nicht nur die Umsetzung eines funktionierenden Modells, sondern vor allem das Verständnis und die Anwendung industrierelevanter Praktiken zur Strukturierung, Automatisierung und Bereitstellung von ML-Systemen.

Konkret verfolgt das Projekt folgende Lernziele:

- **Entwicklung einer modularen ML-Pipeline** unter Verwendung objektorientierter Programmierung in Python für eine bessere Wiederverwendbarkeit und Wartbarkeit.
- **Vertiefung meiner Kenntnisse in Python**, insbesondere im Bereich von Klassen, Modulen, Fehlerbehandlung und Clean Code.
- **Einsatz moderner MLOps-Komponenten**, wie z. B. MLflow für Modell-Tracking, Docker für Containerisierung, S3 als Cloudspeicher, FastAPI für die Bereitstellung eines Vorhersage-Services und CI/CD-Pipeline
- **Umgang mit industriellen Anforderungen an ML-Projekte**, wie Datenvalidierung, Versionierung, reproducible pipelines und automatisiertes Deployment.
- **Aufbau eines skalierbaren und wiederverwendbaren Frameworks**, das über dieses Projekt hinaus auch für andere ML-Anwendungen verwendet werden kann.

Dieses Dokument beschreibt die Struktur und die Funktionalität des Projekts zur Brustkrebs-Klassifikation.

Wichtig: die Programme habe ich nicht zu 100% entwickelt. Das ist natürlich normal, weil es hier vor allem um das Lernen von neuen Technologien und MLOps-Best-Practices geht.

Quelle ist der Online-Kurs in Udemy: Complete MLOps Bootcamp von Krish Naik

Projektstruktur: ETL-Pipeline (Extract – Transform – Load)

Im Zentrum des Projekts steht eine klar strukturierte **ETL-Pipeline**, die es ermöglicht, den gesamten Machine-Learning-Prozess automatisiert, modular und nachvollziehbar abzubilden. Die Pipeline besteht aus den folgenden Schritten:

- **Extract (Datenbeschaffung):** Die Rohdaten werden aus einer externen Quelle geladen (Mongodb) gelesen und in train-test getrennt.
- **Transform (Datenverarbeitung):** Die Daten werden validiert, fehlende Werte mit einem Imputer (KNN) ersetzt, Features standardisiert und in ein für das Modell geeignetes Format transformiert.
- **Load (Speichern & Weiterverarbeitung):** Die verarbeiteten Daten werden als .npy-Dateien gespeichert und stehen den nachfolgenden Modulen wie Modelltraining und Evaluierung zur Verfügung.

Dieser ETL-Ansatz ermöglicht eine klare Trennung der Verantwortlichkeiten und macht die Pipeline robust, wartbar und erweiterbar.

Deployment mit Docker, ECR und EC2

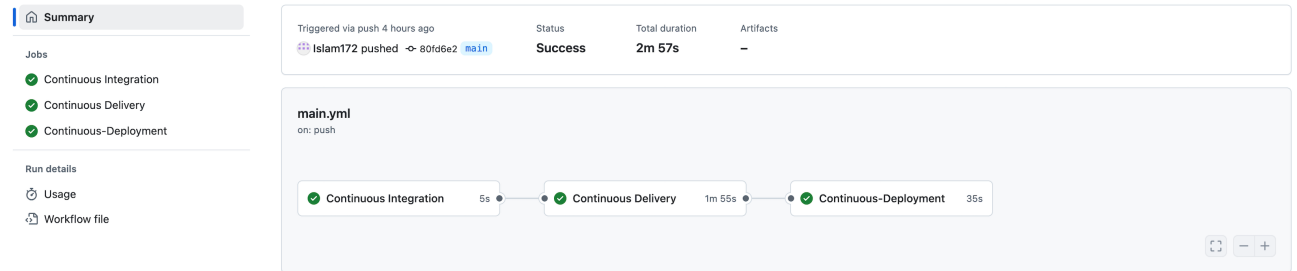
Um das trainierte Modell in einer realistischen Produktionsumgebung bereitstellen zu können, wird ein moderner Deployment-Workflow verwendet:

1. **Docker-Containerisierung:**
 - Das gesamte Projekt (inkl. Modell und Abhängigkeiten) wird in einem leichtgewichtigen Docker-Container verpackt.
 - Dies garantiert eine konsistente Laufzeitumgebung – unabhängig davon, wo das Projekt ausgeführt wird.
2. **Amazon ECR (Elastic Container Registry):**
 - Das erstellte Docker-Image wird in einem privaten Repository in AWS ECR hochgeladen.
 - Dadurch ist das Image zentral versioniert, sicher gespeichert und leicht abrufbar für EC2-Instanzen
3. **Amazon EC2 (Elastic Compute Cloud):**
 - Eine EC2-Instanz dient als Host für die Anwendung.
 - Nach dem Abrufen des Docker-Images aus ECR wird der Container gestartet und stellt über FastAPI einen HTTP-Endpunkt für Vorhersagen zur Verfügung.

Dieser Ansatz stellt sicher, dass das Modell skalierbar, remote zugänglich, leicht wartbar und CI/CD-fähig ist – so wie es in modernen industriellen ML-Projekten gefordert wird.

Projektstruktur: Pfade und Dateien

.github/workflows/: Enthält Konfigurationsdateien für **CI/CD Workflows** via GitHub Actions

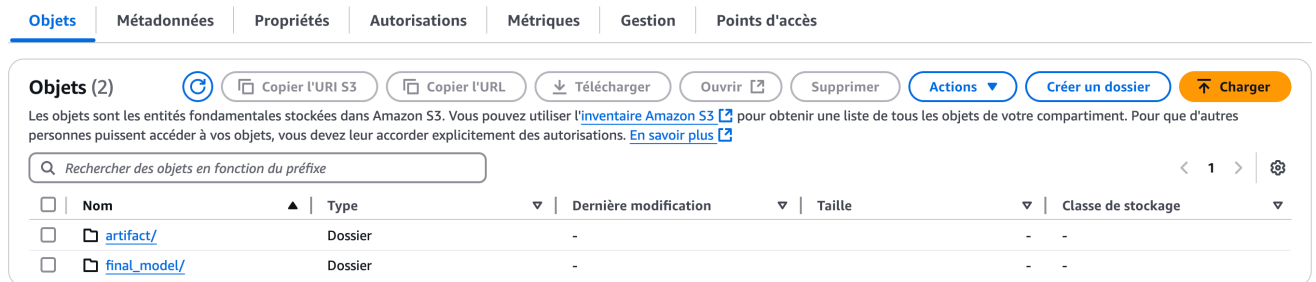


.venv/: Die virtuelle Umgebung mit allen projektbezogenen Python-Paketen.

artifact/: Speichert alle Artefakte, die während der Pipeline erzeugt werden – z. B. trainierte Modelle, transformierte Daten usw. Werden in s3 gespeichert

final_model/: Hier wird das finale trainierte Modell und Preprocessor abgelegt. Wird ebenfalls in s3 gespeichert.

brustkrebs [Info](#)



	Nom	Type	Dernière modification	Taille	Classe de stockage
<input type="checkbox"/>	artifact/	Dossier	-	-	-
<input type="checkbox"/>	final_model/	Dossier	-	-	-

logs/: Speichert automatisch generierte Logs aus dem Trainingsprozess – hilfreich für Debugging und Monitoring.

notebook/: Für explorative Datenanalyse und Visualisierung

prediction_output/: Speichert die vom API-Endpunkt `/predict` generierten CSV-Dateien mit Vorhersagen.

templates/: Beinhaltet HTML-Dateien für die Darstellung in der Weboberfläche (FastAPI). Wird für `table.html` verwendet.

src/: Hauptteil des Projektes

src/cloud/: Operationen für AWS S3

src/components/: Beinhaltet die Modulklassen der ML-Pipeline wie *data_ingestion*, *data_transformation*, *data_validation*, *model_trainer*

src/constants/: Definiert zentrale Konstanten, wie z. B. Pfade und Hyperparameter.

src/entity/: Enthält Entity-Klassen (Konfigurations- oder Artefakt-Datatypes), um die Kommunikation zwischen Komponenten zu standardisieren.

src/exception/: Zentrale Fehlerbehandlung über eigene *CustomException*. Verbessert Debugging und Fehlermeldungen.

src/logging/: Implementiert einheitliches Logging über die gesamte Pipeline

src/pipeline/: Orchestriert die gesamte Trainingspipeline in logischer Reihenfolge.

src/utlis/: Helferfunktionen

Weitere Dateien:

main.py: Test für die Pipeline local vor dem Deployment

app.py: Die FastAPI-Anwendung mit Endpunkten */train* und */predict*. Dient dem Deployment.

file * required
string(binary) test.csv

Responses

Curl

```
curl -X 'POST' \
  "http://127.0.0.1:8080/predict" \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file@test.csv;type=text/csv'
```

Request URL

```
http://127.0.0.1:8080/predict
```

Server response

Code	Details
200	<p>Response body</p> <pre><th>area_worst</th> <th>smoothness_worst</th> <th>compactness_worst</th> <th>concavity_worst</th> <th>concave_points_worst</th> <th>symmetry_worst</th> <th>fractal_dimension_worst</th> <th>prediction</th> </tr> </thead> <tbody> <tr> <th>0</th> <td>13.688</td> <td>16.33</td> <td>87.76</td> <td>575.5</td></pre>

Dockerfile: Definiert das Docker-Image: Installation der Abhängigkeiten und Start der App. Wird in AWS ECR hochgeladen

La vue d'ensemble, l'état et l'ensemble des vulnérabilités de l'analyse des images a été déplacés vers la page détaillée des images. Pour y accéder, cliquez sur une balise d'image.

Images (1)

Rechercher des artefacts

Balise d'image	Type d'artefact	Transmis à	Taille (Mo)	URI de l'image	Digest	Dernière heure d'extraction enregistrée
latest	Image	18 avril 2025, 22:16:17 (UTC+02)	577.23	Copier l'URI	sha256:386820e1d63cf6...	18 avril 2025, 22:18:05 (UTC+02)

push_data.py: Zum manuellen Push von Daten in MongoDB

Atlas ISLAM's Org ... Access Manager Billing All Clusters Get Help ISLAM

Brustkrebs Data Services Charts

Cluster0

Overview Real Time Metrics Collections Atlas Search Query Insights Performance Advisor Online Archive Cmd Line Tools Infrastructure As Code

DATABASES: 1 COLLECTIONS: 1

+ Create Database

Search Namespaces

Dataset

Brustkrebs

Dataset.Brustkrebs

STORAGE SIZE: 200KB LOGICAL DATA SIZE: 436.2KB TOTAL DOCUMENTS: 569 INDEXES TOTAL SIZE: 32KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

Filter Type a query: { field: 'value' }

RESET Apply Options

INSERT DOCUMENT

QUERY RESULTS: 1-20 OF MANY

```
{
  "_id": ObjectId('168012a24aeb239f01e0db63'),
  "id": 842382,
  "diagnosis": "M",
  "radius_mean": 17.99,
  "texture_mean": 10.38,
  "perimeter_mean": 122.8,
  "area_mean": 1001,
  "smoothness_mean": 0.1184,
  "compactness_mean": 0.2776,
  "concavity_mean": 0.3001,
  "concave_points_mean": 0.1471,
  "symmetry_mean": 0.2419,
  "fractal_dimension_mean": 0.07871,
  "radius_se": 1.095,
  "texture_se": 0.9053,
  "perimeter_se": 8.589,
  "area_se": 153.4,
  "smoothness_se": 0.006399,
  "compactness_se": 0.04904,
  "concavity_se": 0.05373
}
```

requirements.txt: Enthält alle Python-Pakete, die für die Ausführung des Projekts benötigt werden.

setup.py: Ermöglicht das Projekt als installierbares Python-Paket.

test_mongodb.py: Skript zum Testen der MongoDB-Verbindung, um sicherzustellen, dass Datenbankzugriffe funktionieren.