**MLOps :**

**Diabetes Prediction**

Bart Dewancker 21/6/2023

Contents

[1. Inleiding 3](#_Toc138228012)

[2. Projectstructuur 3](#_Toc138228013)

[3. Data 4](#_Toc138228014)

[4. Automatisatie van de ML Operations pipeline 5](#_Toc138228015)

[4.1. Azure ML v1 5](#_Toc138228016)

[4.2. Azure ML v2 8](#_Toc138228017)

[5. Applicatie 10](#_Toc138228018)

[6. Versiebeheer 11](#_Toc138228019)

[7. Besluit 14](#_Toc138228020)

# Inleiding

Dit project is een denkbeeldig AI-project voor een huisartsenpraktijk. Met behulp van de applicatie kan een huisarts op basis van enkele gezondheidsgegevens van een patiënt inschatten of de patiënt een verhoogd risico heeft om in de toekomst diabetes te ontwikkelen.

# Projectstructuur

Dit project gebruikt Azure ML v1 voor de data- en modelstappen in de pipeline. De bijbehorende code staat in de scripts- en steps-mappen. Voor de deployment van de pipeline gebruikt het project Kubernetes met helm. De helm-map bevat de benodigde yml-bestanden. De FastApi applicatie staat in de api-map.

De map azureml-2.0-cli bevat de code en yml-bestanden om de data- en modelstappen in de pipeline uit te voeren met Azure ML v2.

A screenshot of a computer

Description automatically generated

Figuur : Projectstructuur.

De automatisatie van de pipeline bestaat uit verschillende stappen die via Github Actions worden uitgevoerd. Het bestand **azure-ml-sdk1.yml** in de map .github beschrijft de stappen met Azure ML v1. Het bestand **azure-ml-sdk2.yml** in dezelfde map beschrijft de stappen met Azure ML v2.

# Data

De dataset met de gezondheidsgegevens komt van de Kaggle website:

<https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset>

Deze dataset bevat één csv-bestand van ongeveer 3,7 MB. Om het project realistischer te maken, bestaat in het project een dataset uit meerdere csv-bestanden, bijvoorbeeld:

* patients\_healthdata\_1.csv
* patients\_healthdata\_2.csv

Figuur 2 toont een voorbeeld van de gegevens in een csv-bestand.

A screenshot of a computer

Description automatically generated with medium confidence

Figuur : Gezondheidsgegevens in een csv-bestand.

Figuur 3 geeft de mogelijke waarden weer voor de velden ‘gender’ en ‘smoking\_history’. De velden ‘hypertension’ en ‘heart\_disease’ hebben het boolean datatype. Voor de andere numerieke velden worden de minimum- en maximumwaarde getoond.

A screenshot of a computer program

Description automatically generated with medium confidence

Figuur : Unieke, minimum en maximum waarden van de gezondheidsgegevens.

# Automatisatie van de ML Operations pipeline

De ML Operations pipeline wordt geautomatiseerd met een Azure Machine Learning (ML) workspace en GitHub Actions. Voor Azure ML kunnen we kiezen tussen versie 1 of versie 2.

## Azure ML v1

Voorbereiding

We hebben in de Azure portal de ML workspace **diabetes-prediction** gemaakt. In de Machine Learning Studio hebben we voor deze workspace de dataset **patients\_healthdata** toegevoegd, die bestaat uit twee bestanden met ruwe gegevens:

* patients\_healthdata\_1.csv,
* patients\_healthdata\_2.csv.

Uitvoering van de pipline

De ML operation pipeline wordt in Github Actions uitgevoerd op een Ubuntu runner. Het bestand **azure-ml-sdk1.yml** beschrijft de jobs die daarbij doorlopen worden. De laatste job, de deployment job, wordt uitgevoerd op een self-hosted runner die gebruik maakt van Kubernetes.

Met inputparameters kunnen we de workflow zo instellen dat bepaalde jobs worden overgeslagen. Als bijvoorbeeld alleen de api opnieuw moet worden gebouwd, kun je de jobs data-preparing en ai-training overslaan.

1. data-preparing

De connectie met Azure wordt eerst gemaakt en de workspace **diabetes-prediction** wordt opgehaald. Daarna wordt de dataset **patients\_healthdata** vanuit Azure naar de runner gehaald. Voor elke bestand in de dataset worden rijen waarin bepaalde gegevens ontbreken of dubbel zijn verwijderd. De gezuiverde bestanden worden lokaal opgeslagen als:

* patients\_healthdata\_1\_checked.csv,
* patients\_healthdata\_2\_checked.csv.

De map met deze bestanden wordt als een nieuwe dataset opgeslagen en geregistreerd in Azure: **patients\_healthdata\_checked**.

Vervolgens wordt deze dataset van Azure naar de runner overgebracht. Elk bestand in de dataset wordt gesplitst in een trainings- en testdeel en opgeslagen in een nieuw bestand, respectievelijk als:

* patients\_healthdata\_1\_train.csv,
* patients\_healthdata\_2\_train.csv,

en

* patients\_healthdata\_1\_test.csv,
* patients\_healthdata\_2\_test.csv.

De mappen met deze bestanden worden als een nieuwe dataset opgeslagen en geregistreerd in Azure, respectievelijke als **patients\_healthdata\_train** en **patients\_healthdata\_test**. Daarna worden alle mappen verwijderd van de runner.

1. ai-training

Deze job begint met het maken van een connectie met Azure, het ophalen van de workspace en het creëren van een compute cluster op Azure. In de workspace wordt een experiment gecreëerd en worden de trainings- en testdataset opgevraagd. Vervolgens wordt op de compute cluster het model gebouwd en begint de trainings- en testfase van het model met een bestand uit elke dataset, bijvoorbeeld respectievelijk:

* patients\_healthdata\_1\_train.csv,
* patients\_healthdata\_1\_test.csv.

Het model bestaat uit een LogisticRegression dat met sklearn in een Pipeline wordt geplaatst. De parameter classifier\_\_C (sterkte van de regularisatie) en classifier\_\_max\_iter van het model kunnen worden aangepast via een omgevingsvariabele die aan de run wordt doorgegeven.

Na de trainings- en testfase worden de metrics van de job opgeslagen in Azure en wordt het model als een pickle-bestand met de naam **pipeline\_logistic\_regression.pkl** op de runner gezet. Daarna wordt de map met dit bestand naar de Azure workspace geüpload en wordt het model geregistreerd onder de naam **diabetes-prediction-model**.

1. api-creation

Deze job verbindt zich met Azure, haalt de workspace op en downloadt het **diabetes-prediction-model** naar de map api/app/outputs op de github runner. Daarna wordt de map api als een artifact met de naam **docker-config** opgeladen in de workflow run, inclusief het model als een pkl-bestand.

1. docker-build-and-push

Deze job begint met het downloaden van de artifact die in de vorige job is opgeladen. De download bevat de api-map met onder andere het pkl-bestand en de docker-compose.yml. Daarna wordt er metadata gegenereerd voor de Docker image op basis van de git-referentie. Deze metadata omvat onder andere de naam van de image, de naam van de git-branch en de git-sha. Met de docker-username en -paswoord uit de git-secrets, wordt er ingelogd op Docker Hub en wordt de image gebouwd en gepusht naar de hub.

1. docker-pull-and-deploy

Deze job is de laatste stap in de pipeline en draait op een self-hosted runner. In de vorige job werd de Docker image getagd met ‘sha-’ gevolgd door zeven cijfers van de git-sha, bijvoorbeeld **sha-e3c0538**. Met deze tag wordt de image uit Docker Hub opgehaald en gedeployed met Kubernetes. Er wordt een namespace gecreëerd in Kubernetes en daarna wordt de deployment uitgevoerd met een helm commando. In deze opdracht worden ook een aantal tags meegegeven, zoals de versie van de app, de versie van het model, de git-sha, de node poort en de replica count.

## Azure ML v2

Voorbereiding

We hebben in de Azure portal de ML workspace **diabetes-prediction-v2** gemaakt. In de Machine Learning Studio hebben we voor deze workspace de dataset **patients\_healthdata** toegevoegd, die bestaat uit twee bestanden met ruwe gegevens:

* patients\_healthdata\_1.csv,
* patients\_healthdata\_2.csv.

Uitvoering van de pipline

Ook in deze workflow kunnen met inputparameters bepaalde jobs worden overgeslagen.

Met Azure ML v2 kunnen we een pipeline op Azure bouwen die uit verschillende componenten bestaat. Elke component heeft in- en/of outputs die de componenten met elkaar verbinden om een pipeline te vormen. Deze pipeline wordt grafisch weergegeven in Azure, bv. zoals te zien in figuur 4.

A screenshot of a computer

Description automatically generated with medium confidence

Figuur : MLops pipeline met Azure ML v2.

Elke component bevat code voor het uitvoeren van zijn specifieke taak. Een job wordt gestart door het eerste component van de pipeline te activeren.

De yml-bestanden voor het bouwen van de componenten en de code voor het uitvoeren van de taken, staan in de map azureml-2.0-cli. We kunnen de componenten bouwen en de pipeline activeren vanuit Azure CLI of vanuit GitHub Actions.

De ML operation pipeline wordt in Github Actions uitgevoerd op een Ubuntu runner. Het bestand **azure-ml-sdk2.yml** beschrijft de jobs die daarbij doorlopen worden. De laatste job, de deployment job, wordt uitgevoerd op een self-hosted runner die gebruik maakt van Kubernetes.

1. az-pipeline

De Service Principal gegevens worden gebruikt om in te loggen op Azure en **diabetes-prediction-v2** workspace wordt ingesteld als de default. Vervolgens worden de vereiste resources en componenten aangemaakt door het bijbehorende yml-bestand aan te roepen:

* az ml compute create --file /azureml-2.0-cli/config/compute.yml --set name=cpu-one
* az ml environment create --file /azureml-2.0-cli/config/diabetes-prediction-env.yml -- version 1.0.0 --set tags.git-sha=<GIT\_SHA>
* az ml component create --file /azureml-2.0-cli/components/dataprep/datacheck.yml -- version 1.0.3 --set tags.git-sha=<GIT\_SHA>
* az ml component create --file /azureml-2.0-cli/components/dataprep/datasplit.yml -- version 1.0.3 --set tags.git-sha=<GIT\_SHA>
* az ml component create --file /azureml-2.0-cli/components/training/training.yml -- version 1.0.3 --set tags.git-sha=<GIT\_SHA>

De laatste opdracht is het starten van een job op Azure met het volgende commando, dat ook een aantal instellingen voor de pipeline bevat:

az ml job create --file /azureml-2.0-cli/pipelines/prediction-pipeline.yml

Deze job levert een getraind model op dat op Azure is geregistreerd.

1. api-creation

Zelfde job als ML v1.

1. docker-build-and-push

Zelfde job als ML v1.

1. docker-pull-and-deploy

Zelfde job als ML v1.

# Applicatie

Verschillende medische labo’s verzamelen en bewaren de gezondheidsgegevens van veel mensen in een databank. In deze databank worden ook persoonsgegevens, zoals geslacht, leeftijd en rookgeschiedenis, opgeslagen met een unieke persoonsID. Zo kan men de gezondheidsgegevens aan een persoon koppelen. De gegevens uit de databank kunnen in een csv-formaat opgeslagen worden om het model te trainen. Het model kan later bijgewerkt worden met meer gezondheidsgegevens van verschillende labo’s.

De applicatie die het model gebruikt, is een api-applicatie gemaakt met FastApi. Met deze applicatie kan een arts via een http-post request voorspellen of een patiënt diabetes kan krijgen op basis van enkele gezondheidsgegevens. Figuur 5 toont een Swagger UI waarmee we de gezondheidsgegevens kunnen invoeren voor een patiënt met id 1.

A screenshot of a computer

Description automatically generated with medium confidence

Figuur : App die gebruik maakt van het diabetes prediction model.

# Versiebeheer

We gebruiken versiebeheer om fouten in de applicatie te kunnen opsporen. Dit kan betrekking hebben op datasets, hyperparameters, code, Docker images, enz.

In de GitHub Actions workflow geven we de versie van de applicatie en het model aan. Deze informatie wordt doorgegeven met de helm-opdracht in de laatste job van de workflow. Ook de git-sha van de laatste commit op de master branch wordt met de helm-opdracht meegegeven. In figuur 6 zien we dat deze metadata in de beschrijving van de recentste Helm Release staan. De naam van de app bevat ook zijn versienummer in zijn extensie, bijvoorbeeld diabetes-prediction**-v1.0.0**.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figuur : De laatste Helm Release bevat informatie over de app-versie en de versie van het model.

Figuur 7 geeft de gegevens van een pod weer waarin staat welke Docker image is gebruikt voor de deployment, bijvoorbeeld: docker-io/bartdewa/diabetes-prediction:sha-e3c0538. Deze gegevens zijn ook te zien in Kubernetes Dashboard.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figuur : Tag van de Docker image die gebruikt is voor de deployment met link naar de git-sha.

Figuur 8 laat zien hoe de git-sha ook in de modelbeschrijving staat in de Azure portal.

A screenshot of a computer

Description automatically generated with medium confidence

Figuur : GIT\_SHA in de modelbeschrijving.

Figuur 9 toont dat de git-sha ook in de metrics van de training te vinden is.

A screenshot of a computer

Description automatically generated

Figuur : git-sha in de metrics van de training.

Figuur 10 toont hoe de code die gebruikt is voor het bewerken van de data ook te traceren is op Github met behulp van de GIT\_SHA.

A screenshot of a computer

Description automatically generated with medium confidence

Figuur : GIT\_SHA van de datasets

Zo kan de versie van elk onderdeel in de volledige pipeline, van de applicatie tot de deployment, modeltraining en datapreparatie, gevolgd worden.

# Besluit

We hebben een webapplicatie ontwikkeld die een huisarts helpt om het risico op diabetes te beoordelen voor een patiënt op basis van enkele gezondheidsgegevens. De applicatie gebruikt een Machine Learning (ML) model om voorspellingen te doen. Om deze applicatie te deployen en te onderhouden, hebben we een pipeline opgezet met verschillende technologieën:

* De webapplicatie gebruikt FastAPI om het model aan te roepen.
* Met Azure v1 of v2 worden de gezondheidsgegevens verwerkt en opgeslagen voor het model en wordt het model getraind en geregistreerd in Azure.
* Met Github Actions wordt de pipeline geautomatiseerd. In de workflow wordt een Docker image gemaakt en naar Dockerhub gestuurd.
* In dezelfde workflow wordt de image van Dockerhub gedownload en gedeployed met Kubernetes. Kubernetes zorgt met replica pods voor een naadloze upgrade van het model zonder dat de gebruiker van de applicatie het merkt.
* Om Kubernetes-applicaties eenvoudiger te implementeren, gebruiken we Helm. Helm stelt ons ook in staat om snel terug te keren naar een vorige implementatie (rollback).

Ten slotte hebben we versienummers en git-sha’s gebruikt om fouten in de applicatie te kunnen opsporen.

Repository op github:

<https://github.com/BartDewancker/mlops-diabetes-prediction>