Documentation of Service Creation for ML Use Cases

Overview

This documentation outlines the steps and procedures followed to create RESTful services for three machine learning (ML) use cases. The use cases are:

1. Clinical Diagnostics Enhancement
2. Research and Development in Oncology
3. Educational and Training Tool for Pathologists

Each use case includes multiple services designed to handle model training, prediction, and evaluation tasks. The services are implemented using Flask for creating API endpoints and Docker for containerizing the application.

ML Use Cases and Services

Clinical Diagnostics Enhancement

1. **Service Name:** DiagnosePatient
    - **Description:** Diagnose a patient based on their medical data.
    - **HTTP Command:** POST
    - **Signature:** https://127.0.0.0:5000/diagnose-patient
    - **Responses:**
        - 200 = success. Diagnosis completed successfully.
        - 400 = error. Invalid request.
        - 500 = error. Internal server error.
    - **Model Example (JSON):**

json

Copy code

```
{
  "patientId": "patient123",
  "patientData": {
    "age": 45,
    "weight": 70,
    "height": 175,
    "symptoms": ["fever", "cough", "fatigue"]
  }
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "patientId": "patient123",
  "diagnosis": [
   {"condition": "Flu", "likelihood": 0.75},
   {"condition": "COVID-19", "likelihood": 0.60}
  ],
  "message": "Diagnosis completed successfully."
}
```

2. **Service Name:** TrainDiagnosticModel
   - **Description:** Train a new version of the diagnostic model with provided training data.
   - **HTTP Command:** POST
   - **Signature:** https://127.0.0.0:5000/train-model
   - **Responses:**
     - 200 = success. Model trained successfully.
     - 400 = error. Invalid request.
     - 500 = error. Internal server error.
   - **Model Example (JSON):**

json

Copy code

```
{
  "trainingDataURL": "https://storage/training-data/diagnostic-data.csv",
  "modelParameters": {
    "learningRate": 0.01,
    "epochs": 10
  },
  "description": "Training data for clinical diagnostic model."
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "status": "Training completed",
  "modelId": "diagnostic_model_v1.2.0",
  "trainingMetrics": {
    "accuracy": 0.94,
    "loss": 0.06
  },
  "message": "The model was successfully trained."
}
```

3. **Service Name:** EvaluateDiagnosticModel
   - **Description:** Evaluate the performance of the diagnostic model with provided evaluation data.
   - **HTTP Command:** POST

- **Signature:** https://127.0.0.0:5000/evaluate-model
- **Responses:**
    - 200 = success. Model evaluated successfully.
    - 400 = error. Invalid request.
    - 500 = error. Internal server error.
- **Model Example (JSON):**

json

Copy code

```
{
  "evaluationDataURL": "https://storage/evaluation-data/diagnostic-evaluation-data.csv",
  "modelId": "diagnostic_model_v1.2.0",
  "evaluationMetrics": ["accuracy", "precision", "recall"]
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "status": "Evaluation completed",
  "modelId": "diagnostic_model_v1.2.0",
  "evaluationResults": {
    "accuracy": 0.93,
    "precision": 0.92,
    "recall": 0.91
  },
  "message": "The model was successfully evaluated."
}
```

Research and Development in Oncology

1. **Service Name:** PredictCancerRisk
    - **Description:** Predict the risk of cancer based on patient genetic and clinical data.
    - **HTTP Command:** POST
    - **Signature:** http://mlops.oncology/predict-cancer-risk
    - **Responses:**
        - 200 = success. Prediction completed successfully.
        - 400 = error. Invalid request.
        - 500 = error. Internal server error.
    - **Model Example (JSON):**

json

Copy code

```
{
  "patientId": "patient456",
  "patientData": {
    "age": 55,
    "geneticMarkers": ["BRCA1", "BRCA2"],
    "lifestyleFactors": ["smoking", "diet"]
  }
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "patientId": "patient456",
  "cancerRisk": {
    "breastCancer": 0.85,
    "lungCancer": 0.60
  },
  "message": "Cancer risk prediction completed successfully."
}
```

2. **Service Name:** TrainOncologyModel
    - **Description:** Train a new version of the oncology model with provided training data.
    - **HTTP Command:** POST
    - **Signature:** http://mlops.oncology/train-model
    - **Responses:**
        - 200 = success. Model trained successfully.
        - 400 = error. Invalid request.
        - 500 = error. Internal server error.
    - **Model Example (JSON):**

json

Copy code

```
{
  "trainingDataURL": "https://storage/training-data/oncology-data.csv",
  "modelParameters": {
    "learningRate": 0.01,
    "epochs": 10
  },
  "description": "Training data for oncology model."
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "status": "Training completed",
  "modelId": "oncology_model_v1.2.0",
  "trainingMetrics": {
    "accuracy": 0.90,
    "loss": 0.10
  },
  "message": "The model was successfully trained."
}
```

3. **Service Name:** EvaluateOncologyModel
   - **Description:** Evaluate the performance of the oncology model with provided evaluation data.
   - **HTTP Command:** POST
   - **Signature:** http://mlops.oncology/evaluate-model
   - **Responses:**
     - 200 = success. Model evaluated successfully.
     - 400 = error. Invalid request.
     - 500 = error. Internal server error.
   - **Model Example (JSON):**

json

Copy code

```
{
  "evaluationDataURL": "https://storage/evaluation-data/oncology-evaluation-data.csv",
```

"modelId": "oncology_model_v1.2.0",

"evaluationMetrics": ["accuracy", "precision", "recall"]

}

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "status": "Evaluation completed",
  "modelId": "oncology_model_v1.2.0",
  "evaluationResults": {
   "accuracy": 0.89,
   "precision": 0.88,
   "recall": 0.87
  },
  "message": "The model was successfully evaluated."
}
```

Educational and Training Tool for Pathologists

1. **Service Name:** AnalyzePathologyImage
   - **Description:** Analyze a pathology image and provide diagnostic insights.
   - **HTTP Command:** POST
   - **Signature:** http://mlops.pathology/analyze-image
   - **Responses:**
     - 200 = success. Image analysis completed successfully.
     - 400 = error. Invalid request.
     - 500 = error. Internal server error.
   - **Model Example (JSON):**

json

Copy code

```
{
  "imageId": "img789",
  "imageURL": "https://storage/pathology-images/image789.jpg"
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
  "imageId": "img789",
  "diagnosticInsights": {
    "cancerCellsDetected": true,
    "cellCount": 250,
    "tumorType": "Adenocarcinoma"
  },
  "message": "Image analysis completed successfully."
}
```

2. **Service Name:** TrainPathologyModel
   - **Description:** Train a new version of the pathology image analysis model with provided training data.
   - **HTTP Command:** POST
   - **Signature:** http://mlops.pathology/train-model
   - **Responses:**
     - 200 = success. Model trained successfully.
     - 400 = error. Invalid request.

        - 500 = error. Internal server error.
    - **Model Example (JSON):**

json

Copy code

```
{
 "trainingDataURL": "https://storage/training-data/pathology-images.zip",
 "modelParameters": {
  "learningRate": 0.01,
  "epochs": 10
 },
 "description": "Training data for pathology image analysis model."
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
 "status": "Training completed",
 "modelId": "pathology_model_v1.2.0",
 "trainingMetrics": {
  "accuracy": 0.92,
  "loss": 0.08
 },
 "message": "The model was successfully trained."
}
```

3. **Service Name:** EvaluatePathologyModel

- **Description:** Evaluate the performance of the pathology image analysis model with provided evaluation data.
- **HTTP Command:** POST
- **Signature:** http://mlops.pathology/evaluate-model
- **Responses:**
    - 200 = success. Model evaluated successfully.
    - 400 = error. Invalid request.
    - 500 = error. Internal server error.
- **Model Example (JSON):**

json

Copy code

```
{
 "evaluationDataURL": "https://storage/evaluation-data/pathology-evaluation-images.zip",
 "modelId": "pathology_model_v1.2.0",
 "evaluationMetrics": ["accuracy", "precision", "recall"]
}
```

- **Example HTTP Response (JSON):**

json

Copy code

```
{
 "status": "Evaluation completed",
 "modelId": "pathology_model_v1.2.0",
 "evaluationResults": {
  "accuracy": 0.91,
  "precision": 0.90,
  "recall": 0.89
 },
 "message": "The model was successfully evaluated."
}
```

API, Flask, and Docker Implementation

1. **API Implementation with Flask:**
   - Flask was used to create RESTful API endpoints for each service. These endpoints handle HTTP POST requests and process the incoming data to perform diagnosis, training, or evaluation tasks.
2. **Flask App Structure:**
   - A basic Flask app structure was created with routes for each service. Each route is defined to handle specific functionalities such as diagnosing a patient, training a model, or evaluating a model.
3. **Docker Implementation:**
   - A Dockerfile was created to containerize the Flask application. This allows the application to be packaged with all its dependencies and run consistently across different environments.
   - **Dockerfile Example:**

dockerfile

Copy code

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app
```

```
# Install Flask

RUN pip install flask


# Make port 5000 available to the world outside this container

EXPOSE 5000


# Define environment variable

ENV NAME ModelTrainingService


# Run app.py when the container launches

CMD ["python", "app.py"]
```

  - The Dockerfile sets up a minimal Python environment, installs Flask, and exposes port 5000 for accessing the API endpoints.

Steps to Run the Dockerized Application

1. **Build the Docker Image:**
    - Navigate to the directory containing the Dockerfile and run the following command:

bash

Copy code

```
docker build -t mlops-service .
```

2. **Run the Docker Container:**
    - Start the container using the following command:

bash

Copy code

```
docker run -p 5000:5000 mlops-service
```

3. **Access the API Endpoints:**

- The services can now be accessed via HTTP requests to http://localhost:5000 using tools like curl or Postman, or programmatically using HTTP client libraries.