Ranya Rhazi

Machine learning Deployment Assignment

Model Deployment Report

Introduction

This report details the development and deployment of a machine learning model using the K-Nearest Neighbors (KNN) algorithm. The model is designed to classify images from the digits dataset, demonstrating the simplicity and effectiveness of KNN for image classification tasks. The application is containerized using Docker and deployed on a cloud server for easy accessibility and scalability.

Architecture Overview

The chosen architecture for this project involves a simple K-Nearest Neighbors (KNN) classifier. KNN is a non-parametric, lazy learning algorithm that classifies new cases based on a similarity measure (e.g., distance functions). KNN has been chosen for its simplicity and effectiveness in classification tasks where the decision boundary is not linear.

Data

The dataset utilized for this project is the digits dataset, available through scikit-learn. This dataset consists of 8x8 pixel images of handwritten digits (0-9), making it suitable for demonstrating image classification. The dataset includes 1,797 samples, which are split into training and testing sets to train and evaluate the model. The features are scaled using MinMax scaling to improve the performance of the KNN algorithm, which is sensitive to the magnitude of data.

Model Development

The KNN model is implemented with scikit-learn using 3 nearest neighbors for its simplicity and effectiveness in classifying the small, relatively uniform images of digits. The model is trained on the scaled pixel values of the digit images, with the target being the actual digit classes ranging from 0 to 9.

Training Process: The model is trained using 75% of the data, with the remaining 25% used for testing and validation.

Evaluation: After training, the model achieves a high accuracy rate on the test set of 98%, demonstrating its efficacy.

Cloud Infrastructure Services

The deployment utilizes a Contabo Linux Ubuntu server to host the containerized application. Docker is employed to containerize the Flask application, ensuring that the application environment is consistent across different systems. The specifics of the deployment process include:

- **Docker**: A Dockerfile is used to create a Docker image that includes the necessary environment, dependencies, and application code.
- **Flask Application**: A lightweight Flask web server is used to provide an API endpoint for making predictions with the model. The server is configured to run on port 5000.

API Configuration

The Flask application exposes a RESTful API endpoint /predict that accepts POST requests containing JSON data with image arrays. The server processes these requests, uses the trained KNN model to predict the digit in the image, and returns the prediction in the response. The workflow is as follows:

Receive JSON Data: The endpoint parses JSON data containing the image.

Preprocessing: The image data is scaled using the same MinMax scaler used during training to match the model's input requirements.

Prediction: The model predicts the class of the digit and the prediction is formatted into a JSON response.

How It Works

To use the API, send a POST request to http://<server-address>:5000/predict with JSON data containing the image array. The server will respond with the predicted digit. Ensure that the image data is in the correct format (flattened 8x8 pixel array) and scaled appropriately if not using the endpoint's scaling.

Testing the Deployment

- **API Testing**: Rigorous tests were conducted to ensure the API handles requests correctly and returns accurate predictions.
- **Performance Monitoring**: The deployment was monitored for performance, including response times and resource usage, to ensure it meets operational standards.

Evaluation and Conclusion

This project demonstrates the practical implementation and deployment of a KNN model for digit classification using Flask and Docker on a cloud server. The simplicity of the model, combined with the robustness of the deployment strategy, makes this a scalable solution for

similar image classification tasks. Future enhancements may include expanding the dataset, experimenting with different algorithms, or extending the API to handle different types of image data.

This setup ensures a seamless integration of machine learning models into production environments, enabling scalable, efficient solutions accessible via standard web protocols.