Task Three: Deploying Machine Learning Models

Overview

Task Three in the D602 course focuses on deploying machine learning models. After developing machine learning pipelines in Task Two using MLflow, the goal in Task Three is to deploy a trained model to a web server. This involves building an API, writing unit tests, creating a Docker image, and deploying it to a cloud environment.

Step-by-Step Deployment Process

1. Export Model Artifacts

For both **R** and **Python**, you'll need to export certain artifacts from Task Two to be used in Task Three:

- **R:** The final model is exported as finalized_models.RDA and the airport encodings as a JSON file.
- **Python:** The final model is exported as a .pkl file, and airport encodings are saved as a JSON file.

These artifacts will be used in your API code to generate predictions.

2. Write a REST API for Model Deployment

The first step in deployment is creating an API to expose the trained model for predictions. A REST API is commonly used for this purpose. The process is as follows:

- **API Overview:** The API will accept HTTP requests through a web server, process the inputs, pass them to the model, and return the predicted result in JSON format.
- **Code Implementation:** Depending on whether you are using R or Python, you will write the API code using the appropriate framework:
    - **R:** Use the plumber package to write the API.
    - **Python:** Use the FastAPI package to create the API.

3. Write Unit Tests for the API

Unit testing is crucial to ensure the functionality and reliability of the deployed API. The unit tests should verify both:

- **Valid input scenarios**: Ensure the model performs as expected for valid inputs.
- **Invalid input scenarios**: Check how the system handles invalid inputs, ensuring that it behaves robustly under error conditions.
- **Testing Frameworks:**
    - **R:** Use the testthat package for writing unit tests.
    - **Python:** Use the pytest package to write and execute unit tests.

4. Bundle API Code into a Docker Image

After creating the API and writing the unit tests, the next step is to bundle the entire application into a Docker image. You do not need to create the image, you only need to write the dockerfile and push it along with the API code and the unit test code. A Docker image encapsulates the code and all its dependencies, making it portable and ready for deployment on various platforms (local or cloud).

- **Dockerfile Creation:** Write a Dockerfile that defines how to build the Docker image, including installing necessary dependencies and running the API server.

Example Dockerfile for Python:

```
FROM python:3.9

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Example Dockerfile for R:

```
FROM rocker/r-ver:4.0.2

WORKDIR /app

COPY . /app

RUN Rscript -e "install.packages('plumber')"
RUN Rscript -e "install.packages('jsonlite')"

CMD ["Rscript", "api.R"]
```

5. Push Docker file, unit test, and API code to GitLab

Once the Docker file is created, push it to GitLab

6. Automate the Process with GitLab CI/CD [This step is done automatically at GitLab. You do not to do anything at this step]

GitLab provides built-in automation to test, build, and push your code to a container registry. The process is automated via a .gitlab-ci.yml file, which orchestrates the pipeline.

- **Steps in the GitLab CI/CD pipeline:**
  - **Testing Stage:** The GitLab CI/CD pipeline will run unit tests (either R or Python, depending on your code submission).
  - **Build Stage:** After passing tests, the pipeline will build the Docker image using the Dockerfile.
  - **Push Stage:** Finally, the Docker image will be pushed to the GitLab container registry.

You must not change the .gitlab-ci.yml file.

7. Deploying the Docker Image

After pushing the Docker image to the container registry, you can deploy it to any cloud platform or local server. The steps will vary depending on your chosen platform, but the general approach is to pull the Docker image from the registry and run it as a container.

R vs. Python Process Flow

For R Users:

1. **Model Artifacts:** Use the exported .RDA model and the airport encodings JSON file.
2. **API Implementation:** Use the plumber package to implement the REST API.
3. **Unit Testing:** Write unit tests using the testthat package.
4. **Dockerfile Creation:** Write a Dockerfile to package the R API and its dependencies.

5. **Push to GitLab:** Submit your code and push it to the GitLab repository.
6. **CI/CD:** GitLab will run unit tests, build the Docker image, and push it to the container registry.

For Python Users:

1. **Model Artifacts:** Use the exported .pkl model and the airport encodings JSON file.
2. **API Implementation:** Use the FastAPI package to implement the REST API.
3. **Unit Testing:** Write unit tests using the pytest package.
4. **Dockerfile Creation:** Write a Dockerfile for the Python API and its dependencies.
5. **Push to GitLab:** Submit your code and push it to the GitLab repository.
6. **CI/CD:** GitLab will run unit tests, build the Docker image, and push it to the container registry.

What you need to submit

- GitLab Link [Important Details]
- Dockerfile (submitted on Gitlab in 2 versions/submits)
- API code file (submitted on Gitlab in 2 versions/submits)
- Unit test code file (submitted on Gitlab in 2 versions/submits)
- Requirements.txt (submitted on Gitlab)
- The model input Json file (submitted on Gitlab)
- Model pkl file (submitted on Gitlab)
- Report with explanation of what you did (doc)

Helpful Resources

- **FastAPI Documentation:** Learn how to use FastAPI for Python APIs and how to create Dockerfiles for FastAPI applications.
- FastAPI Docker Tips
- **Plumber Documentation:** Learn how to use the plumber package in R for creating REST APIs.
- Plumber hosting tips
- **GitLab CI/CD Pipeline Setup:** Reference the official documentation for setting up GitLab CI/CD pipelines.

- **FAQ:** Common errors and solutions while building your API and Docker image.

Conclusion

Task Three focuses on deploying a machine learning model by creating an API, testing it, containerizing the application with Docker, and automating the deployment using GitLab CI/CD. Follow the outlined steps based on your programming language (R or Python), and refer to the resources provided to ensure smooth progress.

If you encounter any issues during the process, don't hesitate to reach out to your course instructors for assistance.