Module 6: Mini Project-1 Part-A

Bike Rental Prediction

Continuous Integration & Continuous Delivery

For this project, we will build a GitHub Actions workflow to automate model training, testing, package building, api dockerizing, and docker image pushing steps for the bike rental count prediction system. Please refer to Module 6 - AST1 and AST2 for this mini-project.

Part A [Mini-project Session - 22 March 2024]

Step 1: Download project folder in your local system:

1.1 Download the given project folder '*bikeshare_project*' on to your system

Step 2: On your GitHub account, create a new repository:

2.1 Create a new repository to store files related to this mini-project

Step 3: Setup a Cloud development environment: (1 point)

3.1 Setup a cloud development environment, such as *GitHub Codespaces*.
3.2 Authenticate the communication between the Cloud dev environment to your GitHub Account by SSH method. (This step is not required in case of *GitHub Codespaces*)

Step 4: Clone the remote repository in cloud dev environment:

4.1 Clone the remote repository in your cloud environment.
4.2 Add the downloaded project folder to this cloned repository.
4.3 Finally, push the changes into the remote GitHub repository.

Step 5: Run your model training, testing, and package building steps on the Cloud environment: (1 point)

5.1 In the cloud environment, create a virtual environment
5.2 Activate the virtual environment, and install the necessary dependencies
5.3 Execute the "*train_pipeline.py*" script to train the model on bike rental dataset
5.4 Run the "*predict.py*" script to generate predictions

5.5 Run the test cases by executing the "*pytest*" command in the terminal (debug if issue persists)
5.6 Run the "*build*" command to create distributable files (.tar, .whl, etc)
5.7 If the errors persist, debug your code and re-run.

Step 6: Run FastAPI application on your system: (1 point)

6.1 Copy the wheel file (.whl) generated in previous step and paste it inside "*bikeshare_model_api*" directory
6.2 Move into the api folder "*bikeshare_model_api*" and Install fastapi dependencies using "*requirements.txt*" (.whl file will be used to install the functionalities of the model into the FastAPI project)
5.3 From inside the api folder, execute the "*app/main.py*" script to start the application and get its url
6.4 Access the application and test its prediction, then stop the application

Step 7: Dockerize the FastAPI application on your system: (2 points)

[Note: Make sure you have set up Docker to execute docker commands for this step. This is not required in the case of *GitHub Codespaces*, as docker comes pre-installed.**]**
7.1 Create a Dockerfile to dockerize "*bikeshare_model_api*" application
7.2 Exit from virtual environment, and check the docker version to see if docker daemon is running
7.3 Using the Dockerfile, create a docker image
7.4 Using the docker image start a new container, and check if the application is running
7.5 On successful run, push the docker image to DockerHub

Step 8: On your GitHub account, for your new repository, add DockerHub credentials within its Secrets: (1 point)

8.1 Get the access token from your DockerHub account settings
8.2 Add the docker username and access token to the Secrets of your repository

Step 9: Push project files to your remote GitHub repository

9.1 Add the Dockerfile to your cloned repository
[Note: Do not include the below files to the repo:

- cache files (*__pychache__*, etc),
- distributable files (*.whl*, etc), and
- trained model (*.pkl* file)

These need to be generated during the GitHub Actions workflow.**]**

9.2 Finally, push the changes into the remote GitHub repository

Step 10: Create a GitHub Actions workflow to automate the steps for model training, testing, package building, api dockerizing, and pushing the docker image: (4 points)

10.1 Create a GitHub Actions workflow to automate the steps for model training, testing, package building, api dockerizing, and docker image pushing
10.2 Add them as different jobs in the workflow
10.3 Add the jobs to run sequentially (debug if issue persists)
10.4 Add below event trigger to the workflow:

- Run on every push to main branch

10.5 Access the running workflow pipeline
10.6 On successful run, access the pushed docker image on your DockerHub account