MLOps Project Report

iLearnDS

A Pedagogy Tool For Data Science

Team Members:

Uswa Nasir	19I-0534
Mehreen Athar	19I-1712
Adeen Ayub	19I-0553

National University of Computer and Emerging Sciences

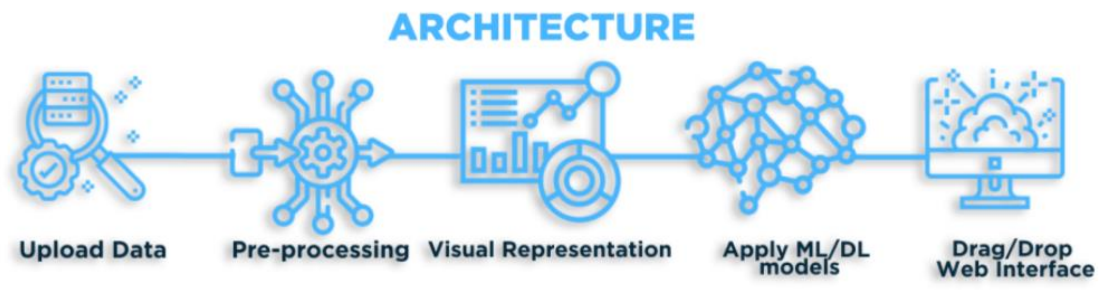
Islamabad, Pakistan

Contents

Introduction

In today’s world data has become a currency, companies are gathering large amounts of data in any shape they can because it is the key to their revenues. Data is the new oil. Data science enables these companies to efficiently understand gigantic data that they have from multiple sources and derive valuable insights to make smarter data-driven decisions. This isn’t the case for everyone. There are a lot of people and smaller companies who have the data but don’t have the required expertise that are needed for understanding data and using it to their advantage. This project is purely for the people who don’t have the necessary mathematical knowledge but want to get a head start in this field. This project is going to provide the audience with an interactive web interface where the user can perform the required tasks without having to actually code. A platform where the user can experiment with different aspects of the data science pipeline and use our wizard to understand the entire workflow.

Architecture



Goals and Objectives

The main problem that we aim to cater to using this project iLearnDS is that we want to make data science accessible to people. Just as there are tools like PowerBI and Tableau that make analysis reporting i.e. visual representation of reports accessible to people using only click-and-drop elements we aim to provide non-tech people a learning platform for the whole data science pipeline that caters to both machine learning and deep learning and all the steps necessary for this in the form of “visual wrappers”. This project will be providing the basic user a platform where the user will be enabled to perform all the necessary steps with ease. This project will not only help the user to get the necessary steps done, it will also be guiding the user as to what the pipeline for its data should be, how the user should approach the data and what can be understood from it

Following are the main objectives of our project:

- **Visual Pedagogy:** iLearnDS web platform's main objective is to be a pedagogy tool, a tool that enables the user to learn how the pipeline of taking data from the stage of preprocessing to applying a model is formed.
- **Interactivity:** Provide visual wrappers to pipeline components in the form of drag-and-drop services.
- **Give Rise to DS:** Give rise to more data scientists so Pakistani industry can flourish and not have to outsource this occupation instead establish and form a strong base of data scientists.

Design Decisions and Chosen Toolset

For the development of the iLearnDS project, several design decisions have been made to achieve the goal of making data science accessible to non-technical users. The chosen toolset includes Git, MLFlow, AirFlow, Jenkins, and Docker, which provide a comprehensive set of tools for version control, experiment tracking, workflow management, continuous integration, and containerization.

- **Git:**

  Git has been chosen as the version control system to manage the source code and track changes throughout the development process. It allows for collaboration among team members, easy branching, and merging of code, ensuring the integrity and traceability of the project.

- **MLFlow:**

  MLFlow has been integrated into the project for experiment tracking and management. It allows the logging of model parameters, metrics, and artifacts during training and evaluation runs. MLFlow's UI and APIs provide a convenient way to track and compare experiments, enabling users to gain insights and analyze the results.

- **AirFlow:**

  Apache AirFlow is utilized as the workflow management tool to orchestrate and automate the data science pipeline. AirFlow enables the scheduling and execution of tasks involved in data preprocessing, model training, evaluation, and deployment. It provides a graphical interface to define and visualize the workflows, making it easier for non-technical users to understand and manage the pipeline.

- **Jenkins:**

  Jenkins is configured as the CI/CD tool for automating testing, building, and deployment processes. It ensures faster and more reliable software delivery by automating repetitive tasks, allowing for continuous integration and deployment. Jenkins integrates with Git to trigger builds and tests whenever changes are pushed to the repository, ensuring the code's quality and stability.

- **Docker:**
  Docker is used for containerization, providing a consistent and isolated environment for the application and its dependencies. By containerizing the application, it becomes portable and can run reliably on different environments, eliminating compatibility issues. Docker simplifies the deployment process and ensures consistency across different deployment targets.

By choosing this toolset, iLearnDS aims to provide a robust and user-friendly platform for non-technical users to learn and explore the data science pipeline. The combination of Git for version control, MLFlow for experiment tracking, AirFlow for workflow management, Jenkins for CI/CD automation, and Docker for containerization ensures efficient development, streamlined workflows, and reliable deployment.

Instructions for running the Dockerized application

1. Open a terminal or command prompt and navigate to the directory containing the Dockerfile and the application code.
2. Build the Docker image by running the command: docker build -t ilearnds-app . (Don't forget the dot at the end).
3. Once the image is built, you can run a container based on the image with the command: docker run -p 5000:5000 ilearnds-app.
4. The iLearnDS application will be accessible at http://localhost:5000 in your web browser.

Guide on reproducing the workflow and experiments

To provide a comprehensive guide on reproducing the workflow and experiments for a GitHub Actions workflow, you can follow these steps:

Prerequisites:

1. Ensure you have a GitHub account.
2. Set up a repository for your project on GitHub.
3. Run the Workflow Locally (Optional):
4. Install and set up the GitHub CLI (command-line interface) tool on your local machine.
5. Use the gh command to authenticate with your GitHub account.
6. Run the workflow locally using the gh command, specifying the event and context.

Monitor Workflow Execution:

1. Go to the "Actions" tab on your GitHub repository.
2. Monitor the workflow runs and their status.
3. View the logs and output of each workflow run to identify any issues or errors.

Reproduce Experiments:

1. Ensure that your codebase is up to date with the desired commit or branch.
2. Trigger the workflow by pushing changes or creating a pull request, depending on the configured triggers.
3. Monitor the workflow run and check the output to ensure the experiments are executed correctly.

Troubleshooting and Debugging:

1. If any issues or errors occur during the workflow execution, review the logs and error messages.
2. Make adjustments to the workflow file, environment setup, or dependencies as needed.
3. Rerun the workflow to test the changes and verify the desired behavior.

Setting up and Configuring DVC

DVC (Data Version Control) is an open-source tool designed to manage and version control data in machine learning and data science projects. It works alongside Git to track changes in datasets, models, and experiments, providing a robust and reproducible workflow. This document will guide you through the setup and configuration of DVC, including the use of Google Drive as the remote storage provider.

1. **Install DVC**
   - Install DVC by running the following command:

     ```
     pip install dvc
     ```

2. **Initialize DVC**
   - Navigate to your project directory in the terminal.
   - Run the following command to initialize DVC in your project:

     ```
     dvc init
     ```

3. **Set Up Git:**
   - Initialize Git in your project directory by running the following command:

     ```
     git init
     ```

4. Add and Commit DVC Files to Git:

- Add the necessary DVC files, including .dvc/ and .dvcignore, to the Git repository.
- Commit the DVC files with an appropriate message using the following commands:

```
git add .dvc/ .dvcignore
git commit -m "Initialize DVC"
```

5. Configure Remote Storage (Google Drive):

- Choose Google Drive as the remote storage provider.
- Configure the remote storage using the following command:

```
dvc remote add -d gdrive://<folder-ID>
```

6. Push Data to Remote Storage:

- Add your data files to DVC using the following command:

```
dvc add <path-to-data-file>
```

- Commit the DVC changes and push the data to the remote storage using the following commands:

```
git add .dvc/ <path-to-data-file>.gitignore

git commit -m "Add data file to DVC"

dvc push
```

7. Pull Data from Remote Storage:

- Retrieve the data files from the remote storage using the following command:

```
dvc pull
```

Conclusion:

By following the steps outlined in this document, you have successfully set up and configured DVC for your project. DVC's integration with Git and remote storage providers like Google Drive ensures efficient version control and reproducibility of your data-centric projects. You can now leverage DVC's powerful features to track changes in data, experiment with different models, and manage your project's development effectively.

Remember to refer to the DVC documentation for more advanced configuration options and to explore additional DVC features tailored to your project's requirements.

Containerization with Docker - Setup and Configuration

Containerization using Docker is a popular approach to package and deploy applications along with their dependencies. Docker provides a consistent environment for running applications across different platforms, making deployment easier and ensuring consistency. This document will guide you through the setup and configuration of Docker for containerizing your application.

1. **Install Docker:**
   - Go to the Docker website (https://www.docker.com/products/docker-desktop) and download the Docker Desktop installer.
   - Double-click on the installer to start the installation process.
   - Follow the on-screen instructions to complete the installation.
   - Once installed, Docker Desktop should be accessible from the system tray.

2. **Create a Dockerfile:**
   - Create a file named Dockerfile in your project directory.
   - Open the Dockerfile and define the instructions to build the Docker image.
   - Include the necessary steps to install dependencies, copy application code, and configure the runtime environment.

3. **Build the Docker Image:**
   - Open a terminal and navigate to your project directory.
   - Run the following command to build the Docker image:

     ```
     docker build -t <image-name> .
     ```

   - Replace <image-name> with the desired name for your Docker image.

4. **Run a Docker Container:**
   - Once the Docker image is built, you can run it as a container.
   - Use the following command to start a Docker container from the image:

     ```
     docker run -d --name <container-name> <image-name>
     ```

   - Replace <container-name> with the desired name for your Docker container and <image-name with the name of the Docker image you built.

5. **Access the Container:**
   - To access the running Docker container, you can use the following command:

     ```
     docker exec -it <container-name> /bin/bash
     ```

- Replace <container-name> with the name of your Docker container.

6. Push Docker Image (Optional):

- If you want to deploy your Docker image to a remote container registry, follow the instructions provided by the registry provider.
- Typically, you need to tag the image with the registry URL and push it using the following commands:

```
docker tag <image-name> <registry-url>/<image-name>

docker push <registry-url>/<image-name>
```

- Replace <image-name> with the name of your Docker image, and <registry-url> with the URL of your container registry.

Conclusion:

By following the steps outlined in this document, you have successfully set up and configured Docker for containerizing your application. Docker provides an efficient and consistent way to package and deploy your application, and its dependencies. You can now leverage the power of Docker to ensure consistent environments across different platforms and simplify your application's deployment process

CI/CD Automation with Jenkins - Setup and Configuration

Jenkins is a popular CI/CD (Continuous Integration/Continuous Deployment) tool used to automate testing, building, and deployment processes. By setting up Jenkins, you can achieve faster and more reliable software delivery through automated pipelines. This document will guide you through the setup and configuration of Jenkins for implementing CI/CD in your project.

1. **Install Jenkins:**
   - Install Jenkins by following the installation instructions for your operating system.
2. **Access Jenkins Web Interface:**
   - Once Jenkins is installed, access the Jenkins web interface by opening a web browser and entering the URL provided during the installation.
3. **Configure Jenkins:**
   - Follow the on-screen instructions to set up the initial configuration of Jenkins, including creating an admin user and specifying the Jenkins URL.
4. **Install Required Plugins:**
   - Install the necessary plugins to enable features and integrations required for your CI/CD pipeline.
   - Navigate to the Jenkins dashboard and click on "Manage Jenkins" -> "Manage Plugins".
   - Install plugins for source code management (e.g., Git), build tools (e.g., Maven), testing frameworks, deployment tools, and any other tools needed for your pipeline.
5. **Configure Jenkins Build Job:**

- Create a new Jenkins job to define the build and deployment process.
- Click on "New Item" on the Jenkins dashboard to create a new job.
- Configure the job details, such as name, description, and type (e.g., Freestyle project or Pipeline).
- Configure the source code management settings, specifying the repository URL, branch, and authentication credentials.
- Define the build steps, which may include compiling code, running tests, and packaging artifacts.
- Configure the post-build actions, such as archiving artifacts or triggering deployment.

6. **Set Up Build Triggers:**

- Configure build triggers to automate the execution of your Jenkins job.
- Specify the conditions that trigger a build, such as changes to the source code repository or a predefined schedule.

7. Configure Deployment:

- Integrate Jenkins with your deployment tools or platforms to automate the deployment process.
- Use plugins or custom scripts to deploy the built artifacts to the target environment, whether it's a staging server, production environment, or a container orchestration platform like Kubernetes.

Conclusion:

By following the steps outlined in this document, you have successfully set up and configured Jenkins as a CI/CD tool for automating testing, building, and deployment processes. Jenkins enables faster and more reliable software delivery through automation, reducing manual effort and ensuring consistent deployment practices. You can now leverage Jenkins to streamline your development workflow, increase efficiency, and achieve continuous integration and deployment of your applications.

Workflow Management with Apache Airflow - Setup and Configuration

Apache Airflow is a powerful workflow management tool that helps orchestrate and automate complex data pipelines, including MLOps workflows. It provides a platform to schedule, monitor, and execute tasks involved in data preprocessing, model training, evaluation, and deployment. This document will guide you through the setup and configuration of Apache Airflow for workflow management in your MLOps pipeline.

1. Install Apache Airflow:

- Install Apache Airflow by following the installation instructions for your operating system.

2. Initialize Airflow Database:

- Initialize the Airflow metadata database by running the following command:

```
airflow db init
```

- Start Airflow Web Server and Scheduler:
- Start the Airflow web server to access the Airflow UI by running the following command:

```
airflow webserver --port <port-number>
```

- Replace <port-number> with the desired port number to access the Airflow UI.
- Start the Airflow scheduler to schedule and execute tasks by running the following command:

```
airflow scheduler
```

3. Configure Airflow DAGs:

- Define your workflow as Directed Acyclic Graphs (DAGs) using Python scripts.
- Create a Python file with the DAG definition and place it in the Airflow DAGs directory (configured in the Airflow settings).

- Define tasks, dependencies, and scheduling intervals in the DAG script.
- Utilize Airflow's operators and hooks to interact with data sources, execute commands, and trigger external systems.

4. Enable DAGs in Airflow:

- By default, Airflow does not automatically detect and enable DAGs. You need to explicitly enable them.
- Run the following command to enable DAGs:

```
airflow dags unpause <dag-id>
```

- Replace <dag-id> with the ID of the DAG you want to enable.

5. Monitor and Manage Workflows:

- Access the Airflow UI by opening a web browser and entering the Airflow web server URL.
- Use the Airflow UI to monitor the status of DAGs and tasks, view logs, and manage workflows.
- Utilize the Airflow CLI commands to manage DAGs, task instances, and variables.

6. Trigger Workflow Execution:

- Manually trigger the execution of a workflow by turning on a DAG in the Airflow UI or using the Airflow CLI.
- Set up triggers based on schedules, external events, or other conditions defined in the DAG.

Conclusion:

configured Apache Airflow as a workflow management tool for your MLOps pipeline. Apache Airflow allows you to orchestrate and automate complex workflows, from data preprocessing to model training, evaluation, and deployment. You can now leverage Airflow's powerful features to streamline your MLOps processes, schedule tasks, monitor workflow execution, and ensure efficient and reliable execution of your machine learning workflows.

Experiment Tracking with MLflow - Setup and Configuration

MLflow is a powerful tool for experiment tracking and management in machine learning projects. It allows you to log model parameters, metrics, and artifacts during training and evaluation runs, enabling easy tracking and comparison of experiments. This document will guide you through the setup and configuration of MLflow to integrate it into your project for experiment tracking.

1. **Install MLflow:**

- Install MLflow by running the following command:

pip install mlflow

2. **Initialize MLflow:**

- Import MLflow in your Python code:

import mlflow

3. **Initialize MLflow in your project:**

mlflow.set_tracking_uri("<mlflow-server-url>")

- Replace <mlflow-server-url> with the URL of your MLflow server. If not using a server, MLflow will use the local file system as a default backend.

4. **Logging Parameters and Metrics:**

- Use the MLflow API to log parameters and metrics during training and evaluation runs.

5. **Start an MLflow run:**

mlflow.start_run()

6. **Log parameters:**

mlflow.log_param("<param-name>", <param-value>)

7. **Log metrics:**

mlflow.log_metric("<metric-name>", <metric-value>)

8. **End the MLflow run:**

   mlflow.end_run()

9. **Logging Artifacts:**

- Use the MLflow API to log artifacts, such as model files, visualizations, or data files.

10. **Start an MLflow run:**

   mlflow.start_run()

11. **Log artifacts:**

   mlflow.log_artifact("<artifact-path>")

- Replace <artifact-path> with the path to the artifact file or directory.

12. **End the MLflow run:**

   mlflow.end_run()

13. **Tracking Experiments:**

- Use the MLflow UI or APIs to track and compare experiments.
- Access the MLflow UI by opening a web browser and entering the MLflow server URL.
- Use the MLflow APIs to query and retrieve information about runs, parameters, metrics, and artifacts programmatically.
-

Conclusion:

By following the steps outlined in this document, you have successfully set up and configured MLflow for experiment tracking and management in your project. MLflow allows you to log model parameters, metrics, and artifacts during training and evaluation runs, facilitating easy tracking and comparison of experiments. You can now leverage MLflow's UI and APIs to explore experiment results, visualize metrics, and track the progress of your machine learning projects.