Week 11 Lab: Git, CI/CD & Automation

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Overview

Objective: Master the "Ops" side of MLOps. You will write tests, automate GitHub workflows, and build an AI-powered code reviewer.

Structure:

1. **Testing**: Write robust unit tests with `pytest`.
2. **CI/CD**: Automate testing with GitHub Actions.
3. **Automation**: Script GitHub interactions with `PyGithub`.
4. **Project**: Build an AI Bot that auto-reviews Pull Requests.

Prerequisites:

- GitHub Account & Personal Access Token (Fine-grained).
- `pip install pytest pytest-cov PyGithub`.

Exercise 1: Testing Basics (45 min)

Goal: Write unit tests for a simple data processing module.

1. **Setup**: Create a folder `my-ml-project/`.
2. **Source Code**: Create `src/utils.py`.

```
def normalize(text):
    if text is None: raise ValueError("Text is None")
    return text.lower().strip()
```

3. **Test Code**: Create `tests/test_utils.py`.

```
import pytest
from src.utils import normalize

def test_normalize_standard():
    assert normalize("  Hello  ") == "hello"

def test_normalize_error():
```

Exercise 2: Testing ML Models (30 min)

Goal: Ensure your model behaves as expected.

1. Create `tests/test_model.py`.
2. Use a **fixture** to create dummy data.

```
@pytest.fixture
def dummy_data():
    return np.random.rand(10, 5), np.random.randint(0, 2, 10)
```

3. Write a test to check if the model overfits on a small batch (sanity check).

```
def test_model_overfits(dummy_data):
    X, y = dummy_data
    model = RandomForestClassifier()
    model.fit(X, y)
    assert model.score(X, y) > 0.9
```

Exercise 3: Setting up CI/CD (45 min)

Goal: Run these tests automatically on every push.

1. Create `.github/workflows/ci.yml`.
2. Define the workflow:

```yaml
name: CI Pipeline
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: {python-version: '3.10'}
      - run: pip install -r requirements.txt pytest
      - run: pytest tests/
```

3. **Push** your code to a new GitHub repo.

Exercise 4: GitHub Automation with Python (30 min)

Goal: Interact with your repo programmatically.

1. Install `PyGithub`.
2. Generate a **Personal Access Token (PAT)** on GitHub.
3. Write a script `auto_issue.py` to create an issue if tests fail locally:

```
from github import Github
import os

# Authenticate
g = Github(os.getenv("GITHUB_TOKEN"))
repo = g.get_user().get_repo("my-ml-project")

# Create Issue
repo.create_issue(
    title="Automated Bug Report",
    body="Tests failed on local machine during nightly run."
)
```

Exercise 5: The AI Code Reviewer (Project) (60 min)

Goal: Build a bot that reviews Pull Requests using an LLM.

Workflow:

1. Script fetches the latest open Pull Request.
2. Gets the file changes (`pr.get_files()`).
3. Extracts the `patch` (diff).
4. Sends the patch to **Gemini/OpenAI API** with prompt:
   "Review this code patch for bugs and style issues."
5. Posts the LLM's response as a comment on the PR.

Deliverable: A working `review_bot.py` that you can run to review your own PRs.

Submission

Submit a link to your GitHub repository containing:

1. `src/` and `tests/` folders (Exercises 1-2).
2. `.github/workflows/ci.yml` (Exercise 3).
3. `review_bot.py` (Exercise 5).
4. A screenshot of the **Actions** tab showing a green build.
5. A screenshot of your **AI Bot's comment** on a PR.

Resources

- **Pytest Docs**: docs.pytest.org
- **GitHub Actions**: docs.github.com/en/actions
- **PyGithub**: pygithub.readthedocs.io