MLPR Prep Assignment - Distance Based Classification

Overview

This assignment will guide you through an end-to-end machine learning workflow while learning key tools like Shell, Git, GitHub, GitHub Actions, Docker, Kaggle, and Weights & Biases (Wandb). You will implement **distance-based classification** using various distance metrics, following best practices for version control, automation, containerisation, and experiment tracking.

P.S. - Feel free to collaborate and work together but don't use LLMs for this assignment please. Its super easy and it's just to introduce you to tools that'll be beneficial while developing your projects. The entire assignment can be done by just looking things up (we've provided links to help you get started). Step wise instructions are provided but we've left some things out so that you get used to solving things on your own.

Each section has its own cheatsheet and write up. In case you don't need it, you can skip directly to the code tasks at the end of each section.

Example Project

To see why all of this is important, check out this link. This project has been made open-source, so people can use the models easily. Even if your project isn't open source, it enables easier collaboration and speeds up development. (Plus now your teammates can't say "but it works on my machine")

Task: Distance-Based Classification

You will apply **distance-based classification** to a dataset, visualise the results, and log your findings. All artefacts (scripts, Docker setup, CI/CD configuration, and experiment logs) must be committed to GitHub.

Section 1: Project Setup (Shell, Git, GitHub)

GitHub Documentation

Why is this important?

Setting up a well-structured project ensures reproducibility and collaboration. Using Git and GitHub allows you to track changes, collaborate with others, and maintain a version-controlled workflow.

Objective:

- Set up a structured project repository.
- Use Git for version control.
- Push initial files to GitHub.

Steps:

1. **Initialise a Git repository**

```
mkdir distance_classification && cd distance_classification
git init

#Alternatively you can create a new repository via the GitHub website

#Then clone your repo
git clone <repo-url>
```

2. **Create a virtual environment**

```
python -m venv venv
source venv/bin/activate  # On Windows, use `venv\Scripts\activate`
```

3. **Install dependencies**

```
pip install numpy pandas matplotlib scikit-learn wandb
```

4. **Create a `.gitignore` file** (ignore `venv`, dataset files, and logs)
5. **Write an initial `distance_classification.py` script** that loads the dataset.
6. **Commit and push to GitHub**

```
git add .
git commit -m "Initial project setup"
git branch -M main
git remote add origin <your-repo-url>
git push -u origin main
```

Code Task:

- Add all the files necessary for the assignment to your repository.
- Make sure you upload the files via CLI or the GitHub extension in your IDE (Don't use the web upload option).

Section 2: Automate Workflow (GitHub Actions)

Github Actions Documentation

Why is this important?

Automation ensures consistency and reduces human error. GitHub Actions allows you to set up continuous integration workflows to automatically test and validate code upon changes. Its also great for automatically fetching live updates from APIs, self-updating datasets and other services.

Objective:

- Set up GitHub Actions to ensure reproducibility.

- Run a basic test script upon each push.

Steps:

1. **Create a GitHub Actions workflow**:
   - In your repository, navigate to `Settings > Actions` and enable workflows if needed.
   - Create a `.github/workflows/main.yml` file with the following content:

```
name: Run Tests
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - name: Install dependencies
        run: |
          pip install numpy pandas scikit-learn
      - name: Run test script
        run: python test_script.py
```

2. **Commit and push this file** to trigger a workflow.
3. **Check GitHub Actions tab** in your repository to see the job status.

Code Task:

- Write a script to verify that all your images and files are loading properly in your Jupyter Notebook.

Section 3: Containerisation (Docker)

Docker Documentation

Why is this important?

Docker ensures that your code runs consistently across different machines by packaging dependencies into a container. This allows your team to all be on the same environment and ensures that dependancy or version related issues do not occur.

Objective:

- Package your environment to ensure consistency across machines.

Steps:

1. **Create a `Dockerfile`** :

```
FROM python:3.11
WORKDIR /app
COPY . .
RUN pip install numpy pandas scikit-learn wandb
CMD ["python", "distance_classification.py"]
```

2. **Build and run the container**:

```
docker build -t distance_classification .
docker run distance_classification
```

3. **Push the Dockerfile to GitHub**

Code Task:

- Edit the Dockerfile to match your environment and push it to GitHub.

Section 4: Experimentation (Kaggle)

Kaggle Documentation

Why is this important?

Kaggle provides cloud-based Jupyter Notebooks with GPU support, allowing you to run experiments efficiently without needing local hardware resources.

Objective:

- Use a Kaggle Notebook to run experiments.

Steps:

1. Create a Kaggle account and start a new notebook. (You'll have to verify your phone number to access the internet and GPUs)
2. Install dependencies:

```
!pip install numpy pandas scikit-learn wandb
```

3. Define Wandb API key in your Kaggle Notebook:

```
import wandb
wandb.login(key="your-wandb-api-key")
```

4. Work on the provided lab assignment and complete the task in the Jupyter Notebook.
5. Save output plots and log findings.
6. Push updated scripts to GitHub.

Code Task:

1. Upload the given assignment material to Kaggle. You can import the Jupyter Notebook directly.
2. **Complete the code:** The skeleton has been provided to you.
3. **Visualise results**: Generate plots for performance comparison.
4. **Save and export final results**: Save key metrics and plots (if any).

Section 5: Tracking Results (Weights & Biases)

WandB Documentation

Why is this important?

Experiment tracking helps monitor performance over multiple runs, making it easier to compare models and fine-tune hyperparameters.

Objective:

- Log metrics and visualise results using Wandb.

Steps:

1. **Sign up for Wandb** at wandb.ai.
2. **Get your API key**:
   - Navigate to Settings on Wandb.
   - Copy your API key.
3. **Log results from Kaggle Notebook**:

```
wandb.init(project='distance_classification_project')

#Example Metrics
wandb.log({"Metric Score": score})
wandb.log({"Output Image": wandb.Image(image)})
```

4. **Run and track multiple experiments.**

Code Task:

- Log your findings, graphs, plots etc to wandb. Add a screenshot of your wandb project dashboard to the `README.md`
- Write a GitHub Action script that automatically reruns your Jupyter Notebook whenever the code is modified. Ensure that the metrics/images are also logged to wandb automatically.
  - Note - You can either modify the previous action script you made or create a new one.

Section 6: Final Submission

Markdown Documentation

Submission:

- Share your **GitHub repository link**. The repo should have the following
  - Your Completed Jupyter Notebook
  - Your Dockerfile
  - The Action Script to rerun your code
- Ensure you've completed the code tasks for each sections
- Include a **report (Markdown)** summarising your findings by adding it to the `README.md`
- Include any images, graphs or other information to support your findings in the readme file too.
