MLOps Lab Exercises

Q1: Git Basics for ML Projects

Question: Initialize a Git repository for a new ML project. Add a Python script, configure `.gitignore` to exclude virtual environments and large files, and push to a remote repository.

Solution Guide:

Create a Hello_world Python file.

```

git init

git add hello_world.py

git commit -m "Initial commit with hello world python script"

echo "venv/" >> .gitignore

echo "*.pyc" >> .gitignore

echo "data/*.csv" >> .gitignore

git add .gitignore

git commit -m "Add .gitignore"

git push [remote repo URL]

```

Q2: Track Dataset Using DVC

Question: Initialize DVC, track a dataset file, commit changes, and push to GitHub.

Prerequisites:

- DVC installed
- A dataset file (e.g., iris_data.csv)

Solution Guide:

```

git init

dvc init
```

dvc add iris_data.csv

git add iris_data.csv.dvc .gitignore .dvc/

git commit -m "Track dataset with DVC"

git push origin main

```

### Q3: Train a Model and Commit

**Question:** Train a simple ML model (e.g., logistic regression) and commit the training script to Git.

- **Solution Guide:**
  - # Save the script as train_model.py
  - from sklearn.datasets import load_iris
  - from sklearn.linear_model import LogisticRegression
  - X, y = load_iris(return_X_y=True)
  - clf = LogisticRegression(max_iter=200)
  - clf.fit(X, y)
  - print("Model trained")
  - git add train_model.py
  - git commit -m "Add model training script"

### Q4: Create an Inference API (Without Docker)

**Question:** Write a Python Flask API that serves model predictions.

- **Solution Guide:**
  - # Save as predict_api.py
  - from flask import Flask, request, jsonify
  - import numpy as np
  - from sklearn.linear_model import LogisticRegression
  - from sklearn.datasets import load_iris
  - app = Flask(__name__)
  - X, y = load_iris(return_X_y=True)
  - model = LogisticRegression(max_iter=200).fit(X, y)
  - @app.route('/predict', methods=['POST'])
  - def predict():
  - data = request.json['features']
  - prediction = model.predict([data])
  - return jsonify({'prediction': int(prediction[0])})
  - if __name__ == '__main__':
```

- app.run(debug=True)

Q5: Dockerize Model Inference

Question: Create a Dockerfile to containerize the Flask inference API.

- **Prerequisites:**
  - predict_api.py
  - requirements.txt with Flask and scikit-learn
- **Solution Guide:**
  - # Dockerfile content
  - FROM python:3.9-slim
  - WORKDIR /app
  - COPY requirements.txt .
  - RUN pip install -r requirements.txt
  - COPY . .
  - CMD ["python", "predict_api.py"]
  - # Build and run
  - docker build -t iris-api .
  - docker run -p 5000:5000 iris-api

Q6: Streamlit UI for Model Inference

Question: Create a Streamlit app to input features and display model prediction.

- **Prerequisites:**
  - streamlit, numpy, scikit-learn installed
  - Trained model code
- **Solution Guide:**
  - # Save as app.py
  - import streamlit as st
  - from sklearn.datasets import load_iris
  - from sklearn.linear_model import LogisticRegression
  - import numpy as np
  - iris = load_iris()
  - X, y = iris.data, iris.target
  - model = LogisticRegression(max_iter=200).fit(X, y)
  - st.title("Iris Prediction")
  - inputs = [st.slider(label, min_value=val[0], max_value=val[1], value=val[2]) for label, val in zip(

- ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'], [(4.0, 8.0, 5.1), (2.0, 4.5, 3.5), (1.0, 7.0, 1.4), (0.1, 2.5, 0.2)])]
- if st.button('Predict'):
- result = model.predict([inputs])[0]
- st.success(f"Prediction: {iris.target_names[result]}")

Q7: Write NGINX Config for Load Balancing

Question: Write an NGINX config to load balance traffic to multiple instances of your ML API running on different ports.

- **Prerequisites:**
  - Multiple instances of model inference app running locally (e.g., ports 5000, 5001)
- **Solution Guide:**
  - # nginx.conf
  - http {
  - upstream ml_backend {
  - server localhost:5000;
  - server localhost:5001;
  - }
  - server {
  - listen 80;
  - location / {
  - proxy_pass http://ml_backend;
  - }
  - }
  - }

Q8: Simulate Traffic Using Locust

Question: Use Locust to simulate multiple users calling the model inference endpoint.

- **Prerequisites:**
  - locust installed
  - Running inference API
- **Solution Guide:**
  - # Save as locustfile.py
  - from locust import HttpUser, task
  - class MLTest(HttpUser):
  - @task
  - def predict(self):

- self.client.post("/predict", json={"features": [5.1, 3.5, 1.4, 0.2]})
- # Run test
- locust -f locustfile.py --host=http://localhost:5000

Q9: Bias Check & Mitigation in UCI Adult Dataset

Question: Check for bias in predictions from a model trained on UCI Adult dataset. Mitigate it and retrain.

- **Prerequisites:**
  - UCI Adult dataset
  - scikit-learn, pandas installed
- **Solution Guide:**
  - # Load dataset and check sex-based accuracy
  - # Apply reweighting or oversampling to balance classes
  - # Retrain and compare fairness metrics

Q10: Create a Simple GitHub Action for ML Pipeline

Question: Configure a GitHub Actions workflow to install dependencies, run `train_model.py`, and print success message.

- **Prerequisites:**
  - train_model.py
  - requirements.txt
- **Solution Guide:**
  - # .github/workflows/train.yml
  - name: Train ML Model
  - on: [push]
  - jobs:
  - build:
  - runs-on: ubuntu-latest
  - steps:
  - - uses: actions/checkout@v3
  - - name: Set up Python
  - uses: actions/setup-python@v4
  - with:
  - python-version: '3.9'
  - - name: Install dependencies
  - run: pip install -r requirements.txt
  - - name: Run training

- run: python train_model.py