Reproducibility Lab

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Overview

By the end of this lab, you will:

- Set up Python virtual environments
- Create requirements.txt and environment.yml
- Build and run Docker containers
- Dockerize an ML project
- Track experiments with MLflow or Weights & Biases
- Set up reproducible pipelines

Structure:

- Part 1: Environment Management (45 min)
- Part 2: Docker Basics (45 min)
- Part 3: Dockerizing ML Project (60 min)
- Part 4: Experiment Tracking (30 min)

Setup (10 minutes)

Install required tools:

```
# Install Docker Desktop
# Download from: https://www.docker.com/products/docker-desktop

# Verify Docker installation
docker --version
docker run hello-world

# Install additional Python packages
pip install mlflow wandb python-dotenv pyyaml
```

Exercise 1.1: Create Virtual Environment (10 min)

Task: Set up isolated Python environment

```
# Create virtual environment
python -m venv ml-env

# Activate (Linux/Mac)
source ml-env/bin/activate

# Activate (Windows)
ml-env\Scripts\activate

# Verify activation
which python  # Should point to ml-env
python --version

# Install some packages
pip install numpy pandas scikit-learn matplotlib

# Check installed packages
pip list

# Deactivate
deactivate
```

Exercise 1.2: Create requirements.txt (10 min)

Task: Document dependencies

```
# Activate environment
source ml-env/bin/activate

# Install project dependencies
pip install numpy==1.24.3 pandas==2.0.2 scikit-learn==1.2.2 matplotlib==3.7.1

# Export to requirements.txt
pip freeze > requirements.txt

# View file
cat requirements.txt
```

Test in new environment:

```
# Deactivate current
deactivate

# Create new environment
```

Exercise 1.3: Conda Environment (15 min)

Task: Create conda environment with YAML

Create `environment.yml`:

```yaml
name: ml-project
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.10
  - numpy=1.24
  - pandas=2.0
  - scikit-learn=1.2
  - matplotlib=3.7
  - jupyter
  - pip
  - pip:
    - mlflow==2.5.0
```

Use the environment:

Exercise 1.4: Set Random Seeds (10 min)

Task: Make code reproducible

Create `utils/reproducibility.py`:

```
import random
import numpy as np
import os

def set_seed(seed=42):
    """
    Set random seeds for reproducibility.
    """
    random.seed(seed)
    np.random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)

    try:
        import torch
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False
    except ImportError:
        pass

# Test
set_seed(42)
print("Random numbers:", [np.random.randint(0, 100) for _ in range(5)])

# Reset and test again
```

Exercise 2.1: First Dockerfile (15 min)

Task: Create simple Docker image

Create `Dockerfile`:

```
# Use Python 3.10 slim image
FROM python:3.10-slim

# Set working directory
WORKDIR /app

# Copy a simple script
COPY hello.py .

# Run the script
CMD ["python", "hello.py"]
```

Create `hello.py`:

```
print("Hello from Docker!")
print("Python version:", __import__('sys').version)
```

Exercise 2.2: Dockerfile with Dependencies (15 min)

Task: Install Python packages in Docker

Create `requirements.txt`:

```
numpy==1.24.3
pandas==2.0.2
scikit-learn==1.2.2
```

Update `Dockerfile`:

```
FROM python:3.10-slim

WORKDIR /app

# Copy requirements first (for caching)
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt
```

Exercise 2.3: Mount Volumes (15 min)

Task: Share data between host and container

Create `data/input.csv`:

```
x,y
1,2
2,4
3,6
4,8
```

Create `process_data.py`:

```
import pandas as pd

# Read from mounted volume
df = pd.read_csv('/app/data/input.csv')
print("Input data:")
print(df)

# Process data
```

Exercise 3.1: Dockerize ML Training (20 min)

Task: Create complete ML training Docker setup

Create `train.py`:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import joblib
import json

# Load data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=42
)

# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.3f}")

# Save model
joblib.dump(model, '/app/models/model.pkl')

# Save metrics
metrics = {'accuracy': accuracy}
with open('/app/models/metrics.json', 'w') as f:
```

Exercise 3.2: Multi-Stage Dockerfile (15 min)

Task: Optimize Docker image size

Create optimized `Dockerfile`:

```
# Stage 1: Builder
FROM python:3.10 AS builder

WORKDIR /app

COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt

# Stage 2: Runtime
FROM python:3.10-slim

WORKDIR /app

# Copy only necessary files from builder
COPY --from=builder /root/.local /root/.local

# Copy application
COPY train.py .
COPY utils/ ./utils/

# Create directories for outputs
RUN mkdir -p /app/models /app/data

# Update PATH
ENV PATH=/root/.local/bin:$PATH
```

Exercise 3.3: Docker Compose for ML (15 min)

Task: Set up multi-container environment

Create `docker-compose.yml`:

```yaml
version: '3.8'

services:
  training:
    build: .
    volumes:
      - ./data:/app/data
      - ./models:/app/models
    environment:
      - RANDOM_SEED=42
    command: python train.py

  mlflow:
    image: ghcr.io/mlflow/mlflow:latest
    ports:
      - "5000:5000"
    volumes:
      - ./mlruns:/mlflow
    command: mlflow server --host 0.0.0.0 --backend-store-uri /mlflow

  notebook:
    image: jupyter/scipy-notebook:latest
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
```

Exercise 3.4: .dockerignore (10 min)

Task: Exclude unnecessary files

Create `.dockerignore`:

```
__pycache__
*.pyc
*.pyo
*.pyd
.Python
*.so
*.egg
*.egg-info
dist/
build/
.git/
.gitignore
.vscode/
.idea/
*.ipynb_checkpoints
.pytest_cache/
.coverage
htmlcov/
*.log
.env
data/raw/
models/*.pth
notebooks/
*.md
```

Exercise 4.1: MLflow Tracking (15 min)

Task: Track experiments with MLflow

Create `train_with_mlflow.py`:

```
import mlflow
import mlflow.sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

# Start MLflow
mlflow.set_experiment("iris-classification")

# Parameters
params = {
    'n_estimators': 100,
    'max_depth': 5,
    'random_state': 42
}

with mlflow.start_run():
    # Log parameters
    mlflow.log_params(params)

    # Load and split data
    iris = load_iris()
    X_train, X_test, y_train, y_test = train_test_split(
        iris.data, iris.target, test_size=0.2, random_state=42
    )

    # Train
    model = RandomForestClassifier(**params)
    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Log metrics
    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("f1_score", f1)

    # Log model
```

Exercise 4.2: Weights & Biases (15 min)

Task: Track experiments with wandb

```
import wandb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize wandb
wandb.init(
    project="iris-classification",
    config={
        "n_estimators": 100,
        "max_depth": 5,
        "random_state": 42
    }
)

config = wandb.config

# Load data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=config.random_state
)

# Train
model = RandomForestClassifier(
    n_estimators=config.n_estimators,
    max_depth=config.max_depth,
    random_state=config.random_state
)
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Log metrics
wandb.log({"accuracy": accuracy})

# Log confusion matrix
cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', ax=ax)
wandb.log({"confusion_matrix": wandb.Image(fig)})

wandb.finish()
```

Bonus Exercise 1: DVC Setup (20 min)

Task: Version control data and models

```
# Initialize DVC
git init
dvc init

# Add data file
dvc add data/large_dataset.csv

# Commit
git add data/large_dataset.csv.dvc data/.gitignore
git commit -m "Add dataset"

# Configure remote storage (S3, GCS, or local)
dvc remote add -d myremote /tmp/dvc-storage
dvc push

# On another machine, pull data
dvc pull
```

Bonus Exercise 2: Configuration with Hydra (20 min)

Task: Manage configurations elegantly

Install:

```
pip install hydra-core
```

Create `configs/config.yaml`:

```yaml
model:
  name: random_forest
  n_estimators: 100
  max_depth: 5

training:
  test_size: 0.2
  random_seed: 42

data:
```

Bonus Exercise 3: Pre-commit Hooks (20 min)

Task: Automate code quality checks

Install:

```
pip install pre-commit black flake8 mypy
```

Create `.pre-commit-config.yaml`:

```yaml
repos:
  - repo: https://github.com/psf/black
    rev: 23.3.0
    hooks:
      - id: black
        language_version: python3.10

  - repo: https://github.com/pycqa/flake8
    rev: 6.0.0
    hooks:
      - id: flake8
        args: [--max-line-length=88]
```

Project Structure Best Practices

Create organized project:

```
mkdir -p ml-project/{data/{raw,processed,external},models,notebooks,src/{data,models,utils},tests,configs}

tree ml-project/
```

Result:

```
ml-project/
├── data/
│   ├── raw/
│   ├── processed/
│   └── external/
├── models/
├── notebooks/
├── src/
│   ├── __init__.py
│   ├── data/
│   ├── models/
│   └── utils/
```

Deliverables

Submit the following:

1. **GitHub repository** with:
    - Organized project structure
    - requirements.txt or environment.yml
    - Dockerfile and docker-compose.yml
    - .dockerignore and .gitignore
    - README with setup instructions
2. **Docker setup** that:
    - Builds successfully
    - Runs training script
    - Mounts data and model volumes
    - Multi-stage for optimization

Testing Checklist

Before submission:

- [ ] Virtual environment activates correctly
- [ ] All dependencies in requirements.txt
- [ ] Docker image builds without errors
- [ ] Docker container runs successfully
- [ ] Volumes mount correctly
- [ ] Random seeds set for reproducibility
- [ ] Experiment tracking works
- [ ] Code runs on fresh environment
- [ ] README has complete instructions
- [ ] .gitignore excludes sensitive files
- [ ] All tests pass

Common Issues and Solutions

Issue: Docker build fails

- Check Dockerfile syntax
- Ensure files exist before COPY
- Check internet connection for downloads

Issue: Permission denied in container

- Run container as non-root user
- Check volume mount permissions

Issue: Package conflicts

- Use exact versions in requirements.txt
- Clear pip cache
- Try conda instead of pip

Issue: Out of memory

Resources

Documentation:

- Docker: https://docs.docker.com/
- Docker Compose: https://docs.docker.com/compose/
- MLflow: https://mlflow.org/docs/
- Weights & Biases: https://docs.wandb.ai/
- DVC: https://dvc.org/doc

Tools:

- Docker Desktop: https://www.docker.com/products/docker-desktop
- Poetry: https://python-poetry.org/
- Hydra: https://hydra.cc/

Tutorials:

- Docker for Data Science

Excellent Work!

You've mastered reproducible ML development!

Next week: Testing & CI/CD for AI

Questions? Office hours: Tomorrow 3-5 PM