02Lab: Collaborative Machine Learning Project with Git

Objective

This lab is designed to teach students how to collaboratively build and execute a machine learning pipeline while adhering to Git best practices. Students will work in pairs, taking on distinct roles, to preprocess data, train a machine learning model, and evaluate it. The lab emphasizes teamwork, division of responsibilities, and proper use of Git for version control and collaboration.

Learning Outcomes

By the end of this lab, students will:

1. Understand how to collaboratively use Git and GitHub for project development.
2. Develop a machine learning pipeline that includes data preprocessing, model training, and model evaluation.
3. Learn to manage feature branches, commit descriptive changes, and create pull requests for collaborative workflows.
4. Gain experience in dividing tasks and responsibilities effectively.

Roles and Responsibilities

Students will work in pairs, with each student assigned a specific role:

- **Student A: Data Preprocessing**
  - Responsible for preparing the dataset for modeling.
  - Writes the script `data_preprocessing.py`.
  - Ensures that large files (e.g., `cleaned_data.csv`) are excluded from the repository by using `.gitignore`.
- **Student B: Model Training and Evaluation**
  - Responsible for training the machine learning model (`model_training.py`) and evaluating its performance (`model_evaluation.py`).
  - Pushes model files and evaluation results, excluding unnecessary large files from the repository.

Git Workflow Overview

- **Fork and Clone the Repository**: Each student will fork the main repository to their own account, then clone their fork locally.
- **Feature Branches**: Always create a separate branch for each feature or task. Do not work directly on the `main` branch.
- **Descriptive Commits**: Commit changes frequently with clear and descriptive messages.
- **Pull Requests (PRs)**: Use PRs to propose changes from your feature branch to the `main` branch.
- **.gitignore Usage**: Ensure that large files (e.g., datasets, models) are excluded from the repository.

Lab Instructions

1. Repository Setup

1. **Fork the Repository**
    - Using my repository : https://github.com/naziherrahel/Software-Development-Technologies.git Each student must fork the repository to their own GitHub account.
    - **Student A** will fork the original repository, then share the fork link with **Student B** for collaboration.
2. **Clone the Fork**

Clone the forked repository to your local machine using the following command:

```
git clone <forked_repository_url>
cd <repository_folder>
```

    - Confirm the presence of `README.md`, and the following scripts (`preprocessing.py`, `training.py`, and `evaluation.py`).
3. **Pull Latest Changes**

Before beginning work, ensure you have the latest code from the `main` branch:

```
git pull origin main
```

2. Create Feature Branches

Each student must work on their own feature branch:

Student A:

```
git checkout -b feature/data-preprocessing
```

Student B:

```
git checkout -b feature/model-training-evaluation
```

Work exclusively within these branches to complete your assigned tasks.

3. Write and Test the Code

Student A: Data Preprocessing

Open `preprocessing.py` and write the following code:

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

# Load the Iris dataset

def load_data():

iris = load_iris()

data = pd.DataFrame(iris.data, columns=iris.feature_names)

data['target'] = iris.target

return data

# Preprocess the data

def preprocess_data(data):

# Add a new feature: the ratio of sepal to petal length

data['sepal_petal_ratio'] = data['sepal length (cm)'] /
data['petal length (cm)']
```

```
# Standardize features (exclude the 'target' column and
'sepal_petal_ratio')

scaler = StandardScaler()

features = data.drop(columns=['target', 'sepal_petal_ratio']) #
Don't scale the ratio feature

data[features.columns] = scaler.fit_transform(features) # Apply
scaling to only features

# Ensure that the target variable is of type integer (it's
categorical)

data['target'] = data['target'].astype(int)

return data

if __name__ == "__main__":

data = load_data() # Load the Iris dataset

data = preprocess_data(data) # Preprocess the dataset (feature
engineering & scaling)

# Save the preprocessed data to a CSV file (but it will be ignored
by Git)

data.to_csv("cleaned_data.csv", index=False) # This file will be
ignored in Git

print("Preprocessed data saved as 'cleaned_data.csv'.")
```

- Test your script locally:

  ```
  python preprocessing.py
  ```

- Commit your changes:

  ```
  git add preprocessing.py
  git commit -m "Added data preprocessing script"
  git push origin feature/data-preprocessing
  ```

Student B: Model Training

Open `training.py` and write the following code:

```
import joblib

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

from preprocessing import load_data, preprocess_data

# Train and evaluate the model

def train_model(data):

X = data.drop("target", axis=1)

y = data["target"]

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a RandomForestClassifier with hyperparameter tuning

model = RandomForestClassifier(n_estimators=150, max_depth=10,
random_state=42)

model.fit(X_train, y_train)

# Save the trained model

joblib.dump(model, "iris_model.pkl")

print("Model saved as 'iris_model.pkl'.")

# Evaluate the model

predictions = model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)

print(f"Accuracy: {accuracy:.2f}")

return model

if __name__ == "__main__":
```

```
data = load_data()

data = preprocess_data(data)

model = train_model(data)
```

- Test your script locally:

  `python training.py`

- Commit your changes:

  `git add training.py iris_model.pkl`

  `git commit -m "Added model training script"`
  `git push origin feature/model-training-evaluation`

Student B: Model Evaluation

Open `evaluation.py` and write the following code:

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix

import pandas as pd

import joblib

import seaborn as sns

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import time

def evaluate_model(model, data):

# Include the same features as in training (no need to drop 'target'
```

```
again)
X = data.drop("target", axis=1)
y = data["target"]
# Make predictions
predictions = model.predict(X)
# Evaluate performance
acc = accuracy_score(y, predictions)
f1 = f1_score(y, predictions, average='weighted')
cm = confusion_matrix(y, predictions)
# Save confusion matrix as an image file
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
# Generate a unique filename using the current timestamp
filename = f'confusion_matrix_{int(time.time())}.png'
plt.savefig(filename) # Save the plot as an image
plt.close() # Close the plot to avoid unnecessary display
return acc, f1
if __name__ == "__main__":
data = pd.read_csv("cleaned_data.csv") # Read the cleaned data
model = joblib.load("iris_model.pkl") # Load the trained model
acc, f1 = evaluate_model(model, data) # Evaluate the model on the data
print(f"Accuracy: {acc:.2f}")
print(f"F1-Score: {f1:.2f}")
```

- **Test your script locally:**

```
python model_evaluation.py
```

- Commit your changes:

```
git add evaluation.py
git commit -m "Added model evaluation script"
git push origin feature/model-training-evaluation
```

4. Create and Review Pull Requests

- Both students should open pull requests to merge their feature branches into `main`.
- Review each other's code, test locally, and provide constructive feedback.
- Only merge the PRs after thorough testing.

5. Run the Full Pipeline

- Clone the updated repository:

```
git clone <repository_url>
cd <repository_folder>
```

- Install dependencies:

```
pip install -r requirements.txt
```

- Run the scripts in sequence:

```
python preprocessing.py
python training.py
python evaluation.py
```

Submission Instructions

- Ensure all changes are merged into the `main` branch.
- Submit the GitHub repository link to the shared folder for review.

Evaluation Criteria

1. Proper use of Git workflow (feature branches, PRs, commits).
2. Functionality and correctness of scripts.
3. Effective teamwork and division of responsibilities.

Good luck, and happy collaborating!