Subject: Deployment of ML models (Assignment 2)
Team Name: Mlops Squad

Team members

Ravishankar V	Rajesh Lohani	Kingsuk Maity	Ashish Chavan
rsv2284@gmail.com	lohani.rajesh@gmail.com	kingsukiitr@gmail.com	devops.ashishchavan@gmail.com

Assignment 2:

Email Spam Detection. Classify emails as spam or not spam using NLP techniques.

Dataset: SMS Spam Collection (Text Classification)

https://www.kaggle.com/datasets/thedevastator/sms-spam-collection-a-more-diverse-dataset

Task 1:

- Create a ML model.
- Build REST API for best_model_parameter[method to be used GET], prediction[method to be used POST] and training[method to be used POST]

Task 2:

- Use hyper-parameter tuning technique to find the best model and log all experiment run using ML-Flow.
- All the Experiment artifact should be visible in the code github repository

Task 3:

- Create a Docker container for the Backed API and push it to docker hub
- A working Docker Image of your application showcasing both the ML-flow UI and the REST API

Solution:

 # Spam Detection API

This project implements a REST API for classifying emails as spam or not spam using NLP techniques.

Features
- Train multiple models (Decision Tree, Logistic Regression, Random Forest, Multinomail Naive Bayes).
- Hyperparameter tuning with MLflow logging.
- REST API endpoints for model parameters, training, and prediction.

- Dockerized application with MLflow UI.

Setup

1. Project Structure (spam-message-detection):

```
MLOPS_SQUAD/
├── app.py                  # Flask REST API application
├── spam_classifier.py      # Script for training models and hyperparameter tuning
├── Dockerfile              # Dockerfile for containerizing the application
├── requirements.txt        # List of Python dependencies
├── README.md               # Project documentation
├── dataset/                # Folder for dataset
│   └── spam.csv            # Dataset file|
├── models/                 # Folder for trained models
│   └── spam_model.pkl      # Best model saved using pickle
│   └── vectorizer.pkl      # Vectorized model|
├── mlruns/                 # MLflow experiment logs (generated after running mlflow)
```

Clone the repository:

- git clone git@github.com:Rajesh-Lohani/mlops_squad.git
- cd mlops_squad

File Descriptions

1. app.py
   Purpose : Implements the Flask REST API.
   Endpoints :
   /**best_model_parameters** (GET): Returns the best model parameters.
   /**train** (POST): Trains the model with new data.
   /**predict** (POST): Predicts whether a message/email is spam or not.
   Dependencies : Loads the best model (spam_model.pkl) and vectorizer.pkl.

2. spam_classifier.py
   Purpose : Script for training multiple models, performing hyperparameter tuning, and logging experiments with MLflow.
   Outputs :
   Saves the best model as spam_model.pkl.
   Logs experiments in the mlruns/ folder.

3. Dockerfile
   Purpose : Defines the Docker image for the application.

4. requirements.txt
   Purpose : Lists all Python dependencies.

5. README.md
   Purpose : Provides an overview of the project, setup instructions, and usage details.

How to run the application using Docker image – available in docker hub :

1. Pull the docker image using below command :

   docker pull rajeshlohani/mlops_squad:latest

2. Run the Docker Container:

   docker run -p 5000:5000 -p 5001:5001 rajeshlohani/mlops_squad

3. Applications :
   Mlflow UI url : http://localhost:5001/
   REST API url : http://localhost:5000/

4. Endpoints
   a. GET /best_model_parameters: Get the best model's hyperparameters.

      Sample url : http://127.0.0.1:5000/best_model_parameters

      Output :

      ```
      {
        "best_model_parameters": {
          "alpha": 0.1,
          "class_prior": null,
          "fit_prior": true,
          "force_alpha": "warn"
        }
      }
      ```

   b. POST /train: Train the model with new data.

      Sample url : http://127.0.0.1:5000/train
      Input dataset :

      ```
      {
       "sms": ["Free entry in 2 a wkly comp", "Hey there, how are you?", "Win a brand new car!", "Can we meet tomorrow?"],
       "label": ["spam", "ham", "spam", "ham"]
      }
      ```

      Output :

      ```
      {
        "accuracy": 96.0,
        "message": "Model trained successfully"
      }
      ```

c. POST /predict: Predict whether a given SMS is spam or not.

Sample url : http://127.0.0.1:5000/predict

Input body :

```
{
   "sms":"Hello! you won 1000"
}
```

Output :

```
{
   "prediction": "spam"
}
```