MEG Synthetic Data Generator

Overview

The MEG Synthetic Data Generator is a Flask-based web app that simplifies the creation of secure, high-quality synthetic data. By leveraging the Masked Ensemble Generator (MEG) framework, it offers a simple yet powerful interface to upload datasets, train models, and generate synthetic datasets for machine learning and testing purposes.

Key Features:

- **Privacy Preservation:** Protects sensitive data using masking techniques.
- **High Utility:** Generates synthetic data that mirrors real-world patterns.
- **Versatility:** Applicable to multiple domains such as healthcare, finance, and cybersecurity.

Features

1. **Upload Training Data:** Upload your CSV dataset to train the MEG model. This feature simplifies data preparation by ensuring compatibility with the model, streamlining workflows for users with pre-existing datasets.
2. **Load Demo Data:** Use pre-loaded demo data for experimentation.
3. **Train the Model:** Train the model to learn patterns from the data.
4. **Generate Synthetic Data:** Create synthetic datasets by specifying sample size.

Prerequisites

- Python 3.9

Installation

1. Clone the repository:
2. git clone https://github.com/DataBytes-Organisation/Katabatic.git

cd meg-synthetic-data-generator

3. Create and activate a virtual environment (optional):
4. python3 -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

5. Install the dependencies:

pip install -r requirements.txt

Usage

1. Start the Flask application:

python app.py

The Flask application processes uploaded data by saving the files to a designated directory and loading them into memory for analysis. When users upload datasets, the application ensures compatibility and provides feedback on successful uploads or errors. After training the model using the uploaded or demo data, the application uses the trained model to generate synthetic datasets. The generated data is formatted into CSV files and made available for download, completing the workflow from data upload to synthetic data generation.

2. Open your web browser and navigate to:

http://127.0.0.1:5000/

3. Use the interface to:
    - Upload or load datasets.
    - Train the model.
    - Generate synthetic data.

Project Structure

├── app.py # Main Flask application. Contains the core logic for handling requests, processing data, and generating synthetic datasets.

├── requirements.txt # Lists all Python dependencies required to run the project.

└── README.md # Provides documentation for setting up, using, and contributing to the project.

├── app.py # Main Flask application (includes HTML templates)

├── requirements.txt # Python dependencies

└── README.md # Documentation

Dependencies

- Flask
- pandas
- scikit-learn
- katabatic (for MEG utilities)

Future Enhancements

1. Expand support for multimodal datasets (e.g., images, videos): Priority - High. Timeline - Q1 2025. This feature will allow integration of diverse data types, broadening the application's usability.
2. Integrate advanced privacy techniques like differential privacy.
3. Add automated evaluation tools for synthetic data quality.

Flask-based web application for generating synthetic data using the **MEG (Masked Ensemble Generator)** framework. It provides a user-friendly interface for uploading data, training a model, and generating synthetic datasets.

Code Breakdown

1. Imports

The required modules and libraries are imported:

- **Flask**: Core framework for the web application.
- **os**: Handles file paths and directory creation.
- **pandas**: For handling and processing datasets.
- **sklearn.model_selection.train_test_split**: Splits the dataset into training and testing sets.
- **BytesIO**: For handling in-memory CSV file creation.
- **katabatic**: Provides MEG-related utilities (get_demo_data) and model adapter (MegAdapter).

2. Flask App Initialization

python

Copy code

app = Flask(__name__)

app.secret_key = "supersecretkey"

- app: Initializes the Flask app.
- secret_key: Enables session management and flash messaging.

3. Global Variables

python

Copy code

training_data = None

adapter = None

model_trained = False

UPLOAD_FOLDER = "uploads"

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

- training_data: Stores the dataset uploaded or loaded.
- adapter: Instance of the MEG model.
- model_trained: Boolean to track if the model has been trained.
- UPLOAD_FOLDER: Directory for saving uploaded files.

4. HTML Template

HTML_TEMPLATE contains the frontend for the application, with Bootstrap for styling. It provides:

- Forms for uploading data, loading demo data, training the model, and generating synthetic data.
- Dynamic messages using Flask's flash mechanism to indicate success or errors.

5. Routes

/ - Home Page

@app.route("/", methods=["GET"])

def index():

...

- Renders the home page using HTML_TEMPLATE.
- Passes context variables (training_data and model_trained) to dynamically show/hide sections.

/upload - Upload Training Data

@app.route("/upload", methods=["POST"])

def upload():

...

- Handles CSV file uploads.
- Saves the file to UPLOAD_FOLDER and reads it into training_data using Pandas.
- Provides feedback via flash messages.

/load_demo - Load Demo Data

@app.route("/load_demo", methods=["POST"])

def load_demo():

...

- Loads a predefined dataset (adult-raw) using get_demo_data.
- Updates training_data and flashes success or error messages.

/train - Train the Model

@app.route("/train", methods=["POST"])

def train():

...

- Splits training_data into features (X) and labels (y), and further into training and testing sets.
- Initializes the MegAdapter instance and trains the model (adapter.fit).
- Updates model_trained to True upon successful training.

/generate - Generate Synthetic Data

@app.route("/generate", methods=["POST"])

def generate():

...

- Checks if the model has been trained.
- Generates synthetic data of a specified sample size (adapter.generate).
- Converts the synthetic data into a CSV file and serves it for download.

6. Frontend Features

- **Progress Message**: Uses JavaScript to display a "Training in Progress" message.
- **Bootstrap Cards**: Organizes functionality into visually distinct sections.
- **Dynamic Forms**: Sections for uploading data, loading demo data, training the model, and generating synthetic data are shown or hidden based on application state.

7. Flask Application Workflow

1. **Upload/Load Data**: User uploads their own dataset or loads demo data.
2. **Train the Model**: Data is split and the MEG model is trained.
3. **Generate Synthetic Data**: Model generates synthetic data, downloadable as a CSV file.

8. Error Handling

- Exceptions are caught and flashed as error messages.
- Provides user-friendly feedback for issues like missing files or training data.

9. Run the Application

```
if __name__ == "__main__":
  app.run(debug=True)
```

- Starts the Flask application in debug mode, which provides detailed error logs during development.

Key Components

Frontend

- The HTML template dynamically adapts based on the backend state (e.g., showing the "Train Model" section only if training data is available).

Backend

- Flask routes handle user interactions like file uploads, model training, and data generation.
- Data processing is handled using pandas and scikit-learn.

Enhancements

- **Security**: Add input validation and authentication to secure file uploads and application access.
- **Real-time Feedback**: Implement progress indicators for model training and synthetic data generation.
- **Scalability**: Optimize for larger datasets or multiple concurrent users.