Ganblr++ documentation

Overview

Ganblr++ is an advanced generative adversarial network tailored for synthesizing high-quality tabular data with mixed feature types. It introduces sophisticated mechanisms such as:

- **Bayesian Gaussian Mixture**-based discretization
- **Truncation-based sampling** for numerical features
- **Training-on-Synthetic-Testing-on-Real** (TSTR) evaluation for model benchmarking

Ganblr++ is designed for tasks requiring high-fidelity synthetic data generation, making it particularly suitable for sensitive domains such as:

- **Healthcare**
- **Finance**
- **Fraud detection**

Its modular design allows it to handle complex datasets with numerical and categorical features seamlessly.

Installation Steps to Set Up

1. Fork the repository.
2. Clone the repository:

Command: `git clone <enter-your-forked-repo-link> cd Katabatic`

1. Create a virtual environment:

Command: `python -m venv venv`

1. Activate the virtual environment:

- On Windows:

Command: `.\venv\Scripts\Activate`

1. Install dependencies:

Command: `pip install -r requirements.txt`

Key Dependencies:

- **Scikit-learn:** Preprocessing, Gaussian Mixture Model, and evaluation.
- **Pandas:** Data manipulation and analysis.
- **Numpy:** Efficient numerical computations.
- **Scipy:** For truncated normal sampling.
- **Tqdm:** Progress monitoring during sampling.

Architecture and Workflow Main Components

1. **DMM Discretizer:**

- Utilizes Bayesian Gaussian Mixture models for numerical feature discretization.
- Encodes numerical features into ordinal representations, supporting smoother integration with GAN training.

2. GANBLR:

- The core GAN model responsible for generating synthetic tabular data.

3. **Synthetic Sampling:**

- Generates synthetic data by combining ordinal categorical and reconstructed numerical features.

4. **Evaluation:**

- Implements TSTR evaluation to measure the utility of synthetic data in real-world machine learning tasks.

Workflow

1. **Data Preprocessing:**

- Numerical columns are discretized using the DMM Discretizer.

- Categorical data is encoded for GAN training compatibility.

2. **Training:**

- The model is trained using adversarial loss on both numerical and categorical features.

3. **Synthetic Data Generation:**

- Ordinal and numerical data are synthesized separately and combined for high-quality outputs.

4. **Evaluation:**

- Accuracy-based benchmarks validate the synthetic data's effectiveness.

Running the Model Key Methods

- **fit():** Trains the GANBLR++ model on input data.
- **sample():** Generates synthetic data based on learned distributions.
- **evaluate():** Performs TSTR evaluation using logistic regression, random forests, or multi-layer perceptrons.

Usage Example

`from ganblrpp import GANBLRPP`

Ganblr++ Initialization and Workflow

To get started with Ganblr++, follow these steps:

1. Initialize Ganblr++

```
model = GANBLRPP(numerical_columns=[0, 2, 4], random_state=42)
```

2. Train Model

```
X_train, y_train = <your_data>, <your_labels> model.fit(X_train, y_train, k=1, batch_size=64, epochs=20)
```

3. Generate Synthetic Data

```
synthetic_data = model.sample(size=1000) print(synthetic_data)
```

4. Evaluate Model Performance

```
accuracy = model.evaluate(X_test, y_test, model='lr') print("Evaluation Accuracy:", accuracy)
```

Configuration

Configuration File Example (config.json):

```
{"numerical_columns": [0, 2, 4],"random_state": 42,"batch_size": 64,"epochs": 20,"k": 1,"warmup_epochs": 1}
```

Example Workflow

1. **Data Preparation:**
    - Identify **numerical columns**.
    - Prepare **datasets** for training and evaluation.
2. **Model Training:**

```
model.fit(X_train, y_train, k=2, batch_size=32, epochs=10)
```

3. **Synthetic Data Generation:**

```
synthetic_data = model.sample(size=500)
```

4. **Evaluation:**
    - Evaluate synthetic data quality: `accuracy = model.evaluate(X_test, y_test, model='rf')`

Evaluation Metrics

- **TSTR Accuracy:** Measures the performance of synthetic data when used to train ML models and tested on real data.
- **Categorical Matching:** Ensures generated categories match real-world distributions.
- **Visualization:**
    - Feature distributions for numerical and categorical variables.
    - Heatmaps for correlation comparison.

Use Cases Applications

- **Healthcare:** Generate synthetic patient records for safe data sharing.
- **Finance:** Produce synthetic data for fraud detection model training.
- **Retail:** Simulate customer behavior for predictive analytics.

Troubleshooting Common Issues

- **Training Errors:** Ensure numerical columns are accurately specified.
- **Poor Data Quality:** Validate input data preprocessing steps.
- **Evaluation Failures:** Confirm synthetic data dimensions align with test data.

Contribution Guidelines

To contribute:

1. Fork the repository:

```
git fork https://github.com/DataBytes-Organisation/Katabatic.git
```

2. Create a feature branch:

```
git checkout -b feature/ganblrpp
```

3. Submit a pull request for review.

Research Context

Ganblr++ builds on the following concepts:

Key Concepts in GANs

1. **Bayesian Gaussian Mixture Models** for numerical feature discretization.
2. **GAN-based Data Generation** for high-fidelity synthetic datasets.
3. **Evaluation Methodologies** like TSTR for real-world performance benchmarking.

How to Implement Ganplr++ with help from VIDUSHI VAIDEHI: A Step-by-Step Guide to Generate Tabular Synthetic Dataset.**Sent:** Monday, 25 November 2024, 8:18 PM

Plan for Implementing GANs in Ganblr++

The Dockerized implementation ensures:

- **Reproducibility:** Encapsulates all dependencies and environment settings.
- **Portability:** Seamless execution across various platforms supporting Docker.
- **Ease of Deployment:** Simplifies setup and execution with minimal configuration.

Features

Advanced Data Processing

- **Integration:** DMMDiscretizer for Bayesian Gaussian Mixture-based numerical data discretization.
- **Robust Handling:** Categorical and numerical features for GAN training.

Synthetic Data Generation

- **TSTR Implementation:** Training on Synthetic, Testing on Real evaluation for benchmarking synthetic data quality.
- **Flexible Sampling:** Methods to generate datasets of varying sizes.

Dockerization

- **Bundling:** The entire model and its dependencies into a Docker container.
- **Support:** Easy setup and reproducible execution across environments.

Adapter Integration

- **GanblrppAdapter:** Ensures seamless integration with the Katabatic SPI, providing compatibility with other Katabatic components.

Setup Instructions

1. **Clone the Repository:** Clone the project repository to your local system:

```
git clone https://github.com/your-username/ganblrplusplus-docker.git cd ganblrplusplus-docker
```

2. **Build the Docker Image:** Build the Docker image using the provided Dockerfile:

```
docker build -t ganblrplusplus:latest .
```

This will create a Docker image named ganblrplusplus with the latest model version and dependencies.

3. **Run the Docker Container:** Execute the GANBLR++ model:

```
docker run --name ganblrplusplus-container ganblrplusplus:latest
```

4. **Access the Container:** For debugging or interaction, access the container using:

```
docker exec -it ganblrplusplus-container /bin/bash
```

Workflow

1. **Model Initialization:** The GanblrppAdapter initializes the GANBLR++ model by setting numerical column indices and other configurations.

```
from ganblrpp_adapter import GanblrppAdapter

adapter = GanblrppAdapter(model_type="discrete", numerical_columns=["col1", "col2"], random_state=42)

adapter.load_model()
```

2. **Training:** Train the GANBLR++ model using real tabular data:

```
adapter.fit(X_train, y_train, epochs=10, batch_size=64)
```

3. **Synthetic Data Generation:** Generate synthetic data samples for analysis:

Generating Synthetic Data

To generate synthetic data, follow these steps:

1. Evaluation

- Evaluate the synthetic data using **TSTR**:

Evaluating Accuracy

Use the following code:

```
accuracy = adapter.evaluate(X_test, y_test, model='lr') print("TSTR Accuracy:", accuracy)
```

Benefits of Dockerization

- Consistency: Avoids dependency conflicts and ensures uniform environments across systems.
- Scalability: Easily deployable on cloud platforms like AWS, GCP, or Azure.
- Streamlined Collaboration: Simplifies sharing of the model with collaborators.

Conclusion

The implementation of **GANBLR++** and its Dockerization enhances the model's accessibility, reproducibility, and usability. By integrating it with the **Katabatic** framework through the **GanblrppAdapter**, it aligns with modern software development practices, making it a robust solution for synthetic tabular data generation.

Implementation Steps

1. Define the Goal

- What to Generate: Identify the type of tabular data relevant to **Ganblr++** (e.g., financial transactions, user activity logs, or predictive features).
- Objective: Ensure synthetic data closely resembles real-world data for testing and model training without compromising sensitive data.

1. Prepare the Dataset

- Real Dataset: Use the data **Ganblr++** operates on (e.g., anonymized user data, transaction logs).
- Preprocessing:
  - Clean and normalize data.
  - Split into features and labels if applicable.
  - Perform exploratory data analysis (EDA) to understand distributions and correlations.

1. Set Up GAN Architecture

- Generator:
  - Create a model to produce synthetic samples with the same structure as the dataset.
  - Use activation functions like **ReLU** and ensure output matches the real data's dimensions.
- Discriminator:
  - Create a model to classify data as real or synthetic.
  - Use **sigmoid** activation in the output layer for binary classification.
- GAN Model:
  - Combine generator and discriminator.
  - Ensure the discriminator's weights are frozen during generator training.

1. Train the GAN

- Use real data and generated samples in each epoch.

- Track losses for both the generator and discriminator to ensure stable training.
- Implement techniques like:
    - Label Smoothing: Avoid overconfident discriminator predictions.
    - Gradient Penalty: Prevent discriminator collapse.

1. Evaluate Synthetic Data

- Model Performance: Train existing models used in **Ganblr++** on synthetic data and compare their accuracy with real data.
- Quality Metrics: Use tools like **table_evaluator** or other statistical measures to evaluate:
    - Similarity between real and synthetic data.
    - Feature distributions and correlations.

1. Integrate and Test in Ganblr++

- Synthetic Data Usage:
    - Test whether synthetic data meets **Ganblr++** requirements (e.g., stress-testing predictive algorithms).
- Visualization:
    - Create plots to compare distributions and highlight areas of improvement.

Next Steps:

1. Dataset Preparation: We will preprocess the relevant **Ganblr++** dataset and split it into features and labels if necessary.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Load your Ganblr++ dataset
data = pd.read_csv('ganblr_dataset.csv')

# Define features and labels (customize for your use case)
features = ['Feature1', 'Feature2', 'Feature3']
label = ['Target']
X = data[features]
y = data[label]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Defining the GAN Architecture

Import necessary libraries:

```
from keras.models import Sequential
from keras.layers import Dense
```

Generator Function

Define the generator:

```
def define_generator(latent_dim, n_outputs):
    model = Sequential()
    model.add(Dense(128, activation='relu', input_dim=latent_dim))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(n_outputs, activation='linear'))
    return model
```

Discriminator Function

Define the discriminator:

```
def define_discriminator(n_inputs):
    model = Sequential()
    model.add(Dense(256, activation='relu', input_dim=n_inputs))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

GAN Function

Define the GAN:

```
def define_gan(generator, discriminator):
    discriminator.trainable = False
    model = Sequential()
    model.add(generator)
    model.add(discriminator)
```

```python
    model.compile(loss='binary_crossentropy', optimizer='adam')
    return model
```

Model Initialization

Initialize models with the following parameters:

- **latent_dim:** 10
- **n_features:** X_train.shape[1]

Create instances:

```python
generator = define_generator(latent_dim, n_features)
discriminator = define_discriminator(n_features)
gan = define_gan(generator, discriminator)
```

Training the GAN

```python
def train_gan(generator, discriminator, gan, X_real, n_epochs=10000, batch_size=64):
    half_batch = int(batch_size / 2)
    for epoch in range(n_epochs):
        # Train discriminator
        idx = np.random.randint(0, X_real.shape[0], half_batch)
        X_real_batch = X_real[idx]
        y_real = np.ones((half_batch, 1))
        d_loss_real = discriminator.train_on_batch(X_real_batch, y_real)

        X_fake = generator.predict(np.random.randn(half_batch, latent_dim))
        y_fake = np.zeros((half_batch, 1))
        d_loss_fake = discriminator.train_on_batch(X_fake, y_fake)

        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # Train generator
        noise = np.random.randn(batch_size, latent_dim)
        y_gan = np.ones((batch_size, 1))
        g_loss = gan.train_on_batch(noise, y_gan)

        # Print progress
        if (epoch + 1) % 1000 == 0:
            print(f"{epoch + 1}/{n_epochs}, d_loss: {d_loss}, g_loss: {g_loss}")

# Train the GAN
train_gan(generator, discriminator, gan, X_train.values)
```

Evaluating the Gan

```python
from table_evaluator import TableEvaluator

# Generate synthetic data
latent_points = np.random.randn(X_train.shape[0], latent_dim)
X_synthetic = generator.predict(latent_points)

# Evaluate
table_evaluator = TableEvaluator(X_train, pd.DataFrame(X_synthetic, columns=features))
table_evaluator.evaluate(target_col=None)

```

GanblrppAdapter Overview

You are probably wondering what the adapter is all about? Let me provide a detailed documentation of it.

The **GanblrppAdapter** is a critical integration component for the **GANBLR++** model, providing compatibility with the **Katabatic** framework through the **KatabaticModelSPI** interface. This adapter encapsulates the GANBLR++ model's functionality, enabling streamlined workflows for data loading, training, and synthetic data generation.

Key Features

- **Simplifies** model initialization and management.
- **Provides** a consistent interface for data processing, training, and evaluation.
- **Handles** edge cases with detailed error handling for robust operations.

Architecture

The GanblrppAdapter is designed to interact seamlessly with the GANBLRPP model by:

1. **Initializing** the model with specified parameters (numerical_columns, random_state).
2. **Loading** datasets directly from CSV files.
3. **Training** the GANBLR++ model with user-specified configurations (epochs, batch_size).
4. **Generating** synthetic datasets while maintaining the structure and distribution of the original data.

Usage and Methods

1. Initialization

The adapter is initialized with essential parameters:

- **model_type:** Specifies the type of model, default is "discrete".
- **numerical_columns:** A list of indices for numerical columns in the dataset.
- **random_state:** Ensures reproducibility by setting the random seed. Example:

```
from ganblrpp_adapter import GanblrppAdapteradapter = GanblrppAdapter(model_type="discrete", numerical_columns=[0, 1], random_state=42)adapter.load_model()
```

2. Data Loading

The **load_data()** method reads datasets from a CSV file and returns a Pandas DataFrame. Example:

```
data = adapter.load_data("path/to/dataset.csv")print(data.head())
```

3. Model Training

The **fit()** method trains the GANBLR++ model using the provided training data. Parameters:

- **X_train:** Features of the training dataset.
- **y_train:** Target variable.
- **k:** Optional GAN parameter, default is 0.
- **epochs:** Number of training epochs, default is 10.
- **batch_size:** Size of training batches, default is 64. Example:

```
adapter.fit(X_train, y_train, k=1, epochs=20, batch_size=32)
```

4. Synthetic Data Generation

The **generate()** method generates synthetic data based on the model's learned distribution. Parameters:

- **size:** Number of samples to generate. Defaults to the training dataset size.

Example:

```
synthetic_data =
adapter.generate(size=100)print(synthetic_data.head())
```

Error Handling

The adapter includes detailed error handling for:

1. **Missing Initialization:** Ensures the model is loaded before training or data generation.
2. **File Loading Errors:** Catches and reports issues with CSV loading.
3. **Training Errors:** Handles inconsistencies during model training. Examples:

- **Uninitialized Model:**

```
RuntimeError: Model is not initialized. Call `load_model()`
first.
```

- **File Not Found:**

Error Notification

[ERROR] An error occurred: **FileNotFoundError**

Integration with Katabatic Framework

The **GanblrppAdapter** adheres to the **KatabaticModelSPI**, ensuring compatibility with other components in the Katabatic ecosystem. This enables a consistent and modular approach for working with various models.

Example Workflow

To use the adapter, follow these steps:

1. Import the necessary libraries:

```
from ganblrpp_adapter import GanblrppAdapterimport pandas as pdimport numpy as np
```

Initialize the Adapter

Set up the adapter with the following code:

```
adapter = GanblrppAdapter(model_type="discrete", numerical_columns=[0, 1], random_state=42)adapter.load_model()
```

Load Dataset

Load your dataset using:

```
data = adapter.load_data("path/to/dataset.csv")X_train = data.drop("target", axis=1)y_train = data["target"]
```

Train the Model

Train the model with:

```
adapter.fit(X_train, y_train, epochs=10, batch_size=64)
```

Generate Synthetic Data

Generate synthetic data using:

```
synthetic_data = adapter.generate(size=50)print(synthetic_data)
```

Updates to GANBLR++ Code

The **GANBLR++** code has undergone significant enhancements to improve robustness, usability, and adherence to best practices. Below is a summary of the key changes made to the old code and their importance.

Key Enhancements

- **Input Validation**
- **Improvement:** Added checks for input_dim and latent_dim to ensure positive integer values.
- **Importance:** Prevents initialization errors and improves robustness.
- **Logging**
- **Improvement:** Introduced logging to monitor training progress and debug effectively.
- **Importance:** Enhances transparency and facilitates troubleshooting during model training.
- **Discriminator Compilation**
- **Improvement:** Explicitly compiled the discriminator with adam optimizer and binary_crossentropy loss.
- **Importance:** Ensures readiness for training and aligns with deep learning best practices.
- **Enhanced Training Process**
- **Improvement:** Implemented a comprehensive training loop:
    - Samples batches of real and fake data.
    - Trains discriminator on both real and fake data.
    - Trains generator to fool the discriminator.
- **Importance:** Produces meaningful results by enabling functional training.
- **Model Saving and Loading**
- **Improvement:** Added methods for saving (save_models) and loading (load_models) models.
- **Importance:** Facilitates reuse of trained models, enhancing usability.
- **Standalone Synthetic Data Generation**

- **Improvement:** Provided a generate_batch method for generating synthetic data independently.
- **Importance:** Increases flexibility in data generation.
- **Configurable Training Parameters**

Key Improvements

- **Improvement:** Made batch size and epochs configurable in the training function.
- **Importance:** Offers greater control over hyperparameters for tailored training.

Why These Changes Matter

- **Improved Reliability:** Input validation and discriminator compilation prevent runtime errors.
- **Enhanced Usability:** Logging, model persistence, and flexible synthetic data generation make the framework more user-friendly.
- **Debugging and Monitoring:** Logging reduces the time required for identifying and resolving issues.
- **Efficient Training:** A structured training loop ensures higher-quality synthetic data.
- **Alignment with Best Practices:** Adhering to deep learning standards ensures broader applicability and better performance.

These updates enhance **GANBLR++'s** functionality, making it a robust solution for high-fidelity synthetic data generation in sensitive domains such as healthcare and finance.

Setting Up the UI and Using Ganblr++

This section guides new users to set up the web interface and use Ganblr++ for generating synthetic data.

1. Prerequisites

- **Flask Application:** Ensure Flask is installed and configured. Use:

```
pip install flask pandas numpy scikit-learn
```

2. Setting Up the UI

HTML Structure: The index.html file includes:

- A **file upload form** for datasets.
- Fields for **training parameters** (epochs and batch size).
- Buttons for starting training and downloading synthetic data.
- A Plotly graph container for visualizing synthetic data.

3. Flask Backend

Essential Endpoints:

- /: Serves the UI (index.html).
- /upload: Accepts and stores uploaded datasets.
- /generate: Processes training, generates synthetic data, and provides a download URL.

Flask Code Overview:

```
app = Flask(__name__)
def preprocess_data(file):
    """Preprocess the uploaded CSV file."""
    try:
        # Load the CSV file into a DataFrame
        data = pd.read_csv(file)
        # Drop the 'Id' column if it exists
```

```
        if 'Id' in data.columns:
            data = data.drop(columns=['Id'])
        # One-hot encode the 'Species' column if it exists
        if 'Species' in data.columns:
            encoder = OneHotEncoder(sparse=False)
            species_encoded = encoder.fit_transform(data[['Species']])
            encoded_columns = encoder.get_feature_names_out(['Species'])
            species_encoded_df = pd.DataFrame(species_encoded, columns=encoded_columns)
            data = pd.concat([data.drop(columns=['Species']), species_encoded_df], axis=1)
    return data
```

Error Handling in Flask

When handling exceptions in Flask, use the following code:

```
except Exception as e:
```

Raise a **ValueError** for preprocessing errors:

```
raise ValueError(f"Error preprocessing the data: {e}")
```

Flask Application Routes

Define your application routes:

- **Index Route:** `@app.route('/') def index(): return render_template('index.html')`
- **Upload Route:** `@app.route('/upload', methods=['POST']) def upload_dataset():`

Upload Dataset Logic

Implement the upload logic:

- Check if **'file'** is in **request.files**:
- Validate the file name:
- Ensure the file is a **CSV**:

Response Handling

Return appropriate responses:

- On success: `return jsonify({'message': 'Dataset uploaded successfully!', 'filepath': filepath}), 200`
- On error: `return jsonify({'error': 'Invalid file format. Please upload a CSV file.'}), 400`

Running the Application

Follow these steps to run the application:

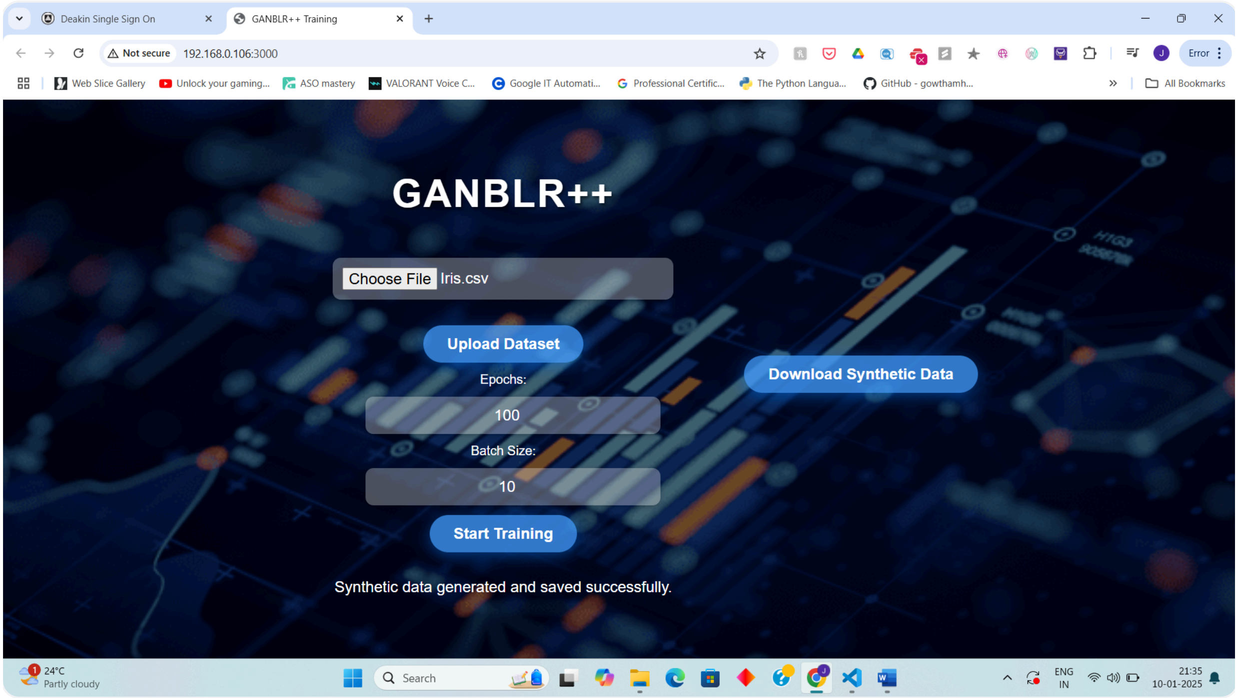
1. Start the Flask app: `python app.py`
2. Open **localhost:3000** in your browser.

Workflow Steps

Follow these steps for the workflow:

1. **Step 1:** Upload a Dataset
    - Choose a **.csv** file from your system and click **Upload Dataset**.
2. **Step 2:** Set Training Parameters

    - Enter values for **Epochs** and **Batch Size**.
3. **Step 3:** Train the Model
    - Click **Start Training** to train the GANBLR++ model.
4. **Step 4:** Download Synthetic Data
    - Upon training completion, a **Download Synthetic Data** button will appear. Click it to download the synthetic dataset.



Conclusion

The **GanblrppAdapter** simplifies the process of integrating the **GANBLR++** model with the **Katabatic** framework, providing a robust and user-friendly interface for managing data workflows. Its modular design and error handling ensure smooth execution across different environments.

References

Key references include:

1. Goodfellow, I., et al. (2014). *Generative Adversarial Networks*. Advances in Neural Information Processing Systems.
   https://arxiv.org/abs/1406.2661
2. Xu, L., et al. (2019). *Modeling Tabular Data using Conditional GAN*. Advances in Neural Information Processing Systems.

https://arxiv.org/abs/1907.00503

3. Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research.

   https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

4. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

   https://link.springer.com/book/10.1007/978-0-387-45528-0

5. Scikit-learn Documentation: Comprehensive guide for Scikit-learn API, covering machine learning and preprocessing tools.

   https://scikit-learn.org/stable/documentation.html

6. Docker Documentation: Official guide for building and running Docker containers.

   https://docs.docker.com/

7. Katabatic Framework Documentation: Detailed documentation for the Katabatic SPI and its integration capabilities.

   https://github.com/DataBytes-Organisation/Katabatic

8. Bayesian Gaussian Mixture Documentation (Scikit-learn): Overview of the Bayesian Gaussian Mixture model used in data discretization.

   https://scikit-learn.org/stable/modules/mixture.html