**Ganblr++ Documentation**

**1. Overview**

**Ganblr++** is an advanced generative adversarial network tailored for synthesizing high-quality tabular data with mixed feature types. It introduces sophisticated mechanisms like Bayesian Gaussian Mixture-based discretization, truncation-based sampling for numerical features, and a 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, and fraud detection. Its modular design allows it to handle complex datasets with numerical and categorical features seamlessly.

**2. Installation**

**Steps to Set Up:**

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

bash

cd Katabatic

1. Create a virtual environment:

bash

python -m venv venv

1. Activate the virtual environment:
   * On Windows:

bash

.\venv\Scripts\Activate

1. Install dependencies:

bash

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.

**3. 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 DMMDiscretizer.
   * 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.

**4. 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***

# Initialize Ganblr++

***model = GANBLRPP(numerical\_columns=[0, 2, 4], random\_state=42)***

# Train model

***X\_train, y\_train = <your\_data>, <your\_labels>***

***model.fit(X\_train, y\_train, k=1, batch\_size=64, epochs=20)***

# Generate synthetic data

***synthetic\_data = model.sample(size=1000)***

***print(synthetic\_data)***

# Evaluate model performance

accuracy = model.evaluate(X\_test, y\_test, model='lr')

print("Evaluation Accuracy:", accuracy)

**5. Configuration**

**Configuration File Example (config.json):**

json

***{***

***"numerical\_columns": [0, 2, 4],***

***"random\_state": 42,***

***"batch\_size": 64,***

***"epochs": 20,***

***"k": 1,***

***"warmup\_epochs": 1***

}

**6. Example Workflow**

1. **Data Preparation:**
   * Identify numerical columns.
   * Prepare datasets for training and evaluation.
2. **Model Training:**
   * Fit the model with specified parameters:

python

model.fit(X\_train, y\_train, k=2, batch\_size=32, epochs=10)

1. **Synthetic Data Generation:**
   * Generate synthetic data:

synthetic\_data = model.sample(size=500)

1. **Evaluation:**
   * Evaluate synthetic data quality:

accuracy = model.evaluate(X\_test, y\_test, model='rf')

**7. Evaluation**

**Metrics:**

* **TSTR Accuracy:** Measures the performance of synthetic data when used to train machine learning 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.

**8. Use Cases**

**Applications:**

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

**9. 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.

**10. Contribution Guidelines**

To contribute:

1. Fork the repository:

bash

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

1. Create a feature branch:

bash

git checkout -b feature/ganblrpp

1. Submit a pull request for review.

**11. Research Context**

**Ganblr++** builds on the following concepts:

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.

**12. References**

1. Goodfellow et al., 2014. "Generative Adversarial Networks."
2. Xu et al., 2019. "Modeling Tabular Data Using Conditional GAN."
3. Scikit-learn Documentation.

**Ganblr++ Documentation**

**1. Overview**

**Ganblr++** is an advanced generative adversarial network tailored for synthesizing high-quality tabular data with mixed feature types. It introduces sophisticated mechanisms like Bayesian Gaussian Mixture-based discretization, truncation-based sampling for numerical features, and a 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, and fraud detection. Its modular design allows it to handle complex datasets with numerical and categorical features seamlessly.

**2. Installation**

**Steps to Set Up:**

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

bash

git clone <enter-your-forked-repo-link>

cd Katabatic

1. Create a virtual environment:

bash

python -m venv venv

1. Activate the virtual environment:
   * On Windows:

bash

.\venv\Scripts\Activate

1. Install dependencies:

bash

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.

**3. 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 DMMDiscretizer.
   * 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.

**4. 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

# Initialize Ganblr++

model = GANBLRPP(numerical\_columns=[0, 2, 4], random\_state=42)

# Train model

X\_train, y\_train = <your\_data>, <your\_labels>

model.fit(X\_train, y\_train, k=1, batch\_size=64, epochs=20)

# Generate synthetic data

synthetic\_data = model.sample(size=1000)

print(synthetic\_data)

# Evaluate model performance

accuracy = model.evaluate(X\_test, y\_test, model='lr')

print("Evaluation Accuracy:", accuracy)

**5. Configuration**

**Configuration File Example (config.json):**

json

{

"numerical\_columns": [0, 2, 4],

"random\_state": 42,

"batch\_size": 64,

"epochs": 20,

"k": 1,

"warmup\_epochs": 1

}

**6. Example Workflow**

1. **Data Preparation:**
   * Identify numerical columns.
   * Prepare datasets for training and evaluation.
2. **Model Training:**
   * Fit the model with specified parameters:

python

model.fit(X\_train, y\_train, k=2, batch\_size=32, epochs=10)

1. **Synthetic Data Generation:**
   * Generate synthetic data:

python

synthetic\_data = model.sample(size=500)

1. **Evaluation:**
   * Evaluate synthetic data quality:

accuracy = model.evaluate(X\_test, y\_test, model='rf')

**7. Evaluation**

**Metrics:**

* **TSTR Accuracy:** Measures the performance of synthetic data when used to train machine learning 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.

**8. Use Cases**

**Applications:**

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

**9. 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.

**10. Contribution Guidelines**

To contribute:

1. Fork the repository:

bash

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

1. Create a feature branch:

bash

git checkout -b feature/ganblrpp

1. Submit a pull request for review.

**11. Research Context**

**Ganblr++** builds on the following concepts:

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 with G...](https://teams.microsoft.com/l/message/19:a65f69a3e19144ab94bf077116128c4a@thread.v2/1732526336130?context=%7B%22contextType%22%3A%22chat%22%7D)

sent on Monday, 25 November 2024 8:18 pm

**Plan for Implementing GANs in Ganblr++**

**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.

**2. 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.

**3. 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.

**4. 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.

**5. 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.

**6. 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.

**Implementation Steps**

**1. Dataset Preparation**

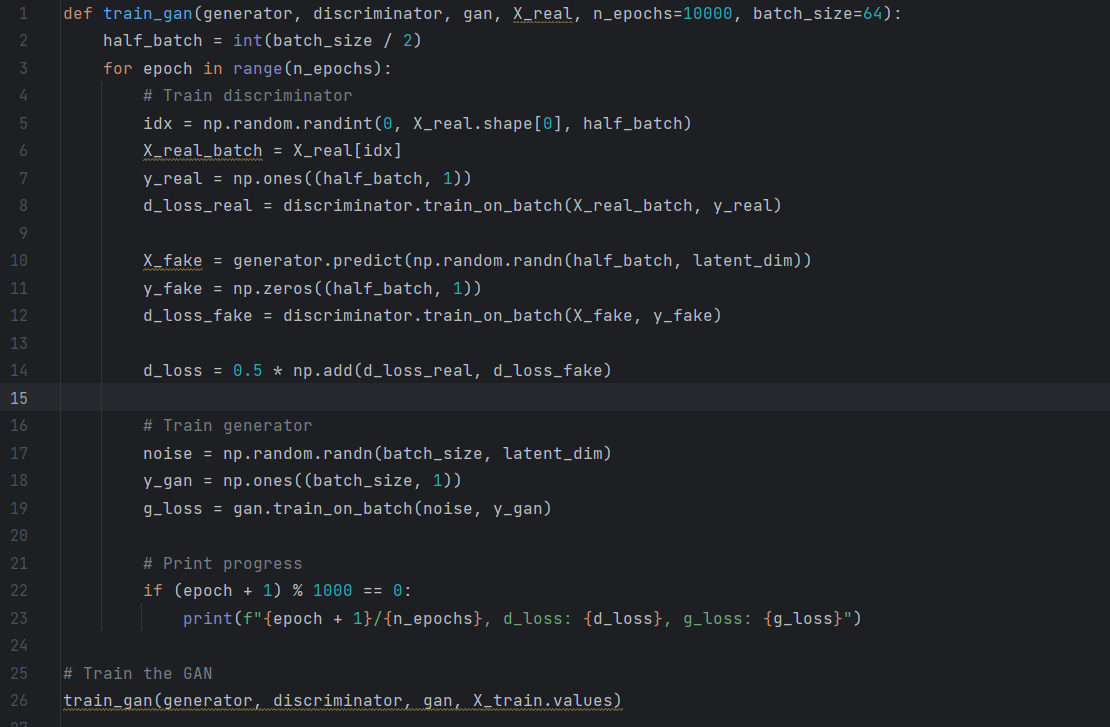
We will preprocess the relevant Ganblr++ dataset and split it into features and labels if necessary.



***Defining the GAN Architecture***

from keras.models import Sequential  
from keras.layers import Dense  
  
# 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  
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  
def define\_gan(generator, discriminator):  
 discriminator.trainable = False  
 model = Sequential()  
 model.add(generator)  
 model.add(discriminator)  
 model.compile(loss='binary\_crossentropy', optimizer='adam')  
 return model  
  
# Initialize models  
latent\_dim = 10  
n\_features = X\_train.shape[1]  
generator = define\_generator(latent\_dim, n\_features)  
discriminator = define\_discriminator(n\_features)  
gan = define\_gan(generator, discriminator)

Training The Gan



Evaluating the Gan

A computer screen shot of a program code

Description automatically generated

**12. References**

1. Goodfellow et al., 2014. "Generative Adversarial Networks."
2. Xu et al., 2019. "Modeling Tabular Data Using Conditional GAN."
3. Scikit-learn Documentation.