QIDLearningLib User Manual: Quasi-Identifier Selection in Tabular Datasets

Sancho Amaral Simões

July 2, 2025

Abstract

This user manual provides comprehensive guidance on utilizing `QIDLearningLib`, a powerful Python library designed for the automated recognition and evaluation of Quasi-Identifiers (QIDs) in tabular datasets. It covers installation, core concepts, detailed usage of optimization algorithms for QID selection, and tips for effective data privacy and utility assessment. This updated version includes instructions on exporting results and a detailed method for metric correlation analysis.

Contents

1 Introduction to `QIDLearningLib`

Quasi-identifiers (QIDs) are attributes in a dataset that, while not directly unique identifiers, can be combined with other information to re-identify individuals, posing significant privacy risks in data sharing and analysis. `QIDLearningLib` is the first Python library specifically developed for automated QID recognition in tabular datasets.

The library integrates various metrics from causality, data privacy, and data utility to offer a holistic assessment of attribute properties and facilitates flexible QID selection. It provides tools for evaluating the performance of QID selection system against a ground truth and includes multiple optimization algorithms for QID selection based on user-defined metrics. Furthermore, `QIDLearningLib` supports redundancy analysis and offers graphical and testing tools for enhanced interpretability.

2 Installation

To get started with `QIDLearningLib`, you need to obtain the library from its official GitHub repository.

2.1 Prerequisites

Ensure you have Python installed on your system.

2.2 Installation Steps

`QIDLearningLib` can be installed by cloning the GitHub repository. The Python package is located within the cloned repository.

1. **Clone the repository:** Open your terminal or command prompt and run the following command:

```
    git clone https://github.com/smartlord7/QIDLearningLib.git
```

2. **Navigate to the library directory:**

```
cd QIDLearningLib/src/QIDLearningLib

```

3. **Integrate into your project:** You can then move this package to your project directory or set up your Python environment to include it.

For more detailed documentation, refer to the documentation available within the GitHub repository at `https://github.com/smartlord7/QIDLearningLib.git`.

3 Core Concepts

`QIDLearningLib` revolves around several core concepts crucial for effective QID selection.

3.1 Quasi-Identifiers (QIDs)

QIDs are attributes (e.g., age, gender, zipcode) that, while not directly unique identifiers, can be combined with other information to re-identify individuals in a dataset. The primary goal of this library is to help identify an optimal set of QIDs that balances privacy risks with data utility.

3.2 Metrics for QID Assessment

The library integrates a diverse set of metrics to quantify various aspects of QIDs and their impact on data privacy and utility. These metrics are crucial for evaluating QID selection.

3.2.1 Causality Metrics

These metrics are designed to identify attributes that influence "treatment effects" or other sensitive attributes, making them potential QID candidates.

- **Covariate Shift (CS):** Quantifies disparities in QID distribution across different groups. A high CS suggests strong QID impact.
- **Balance Test:** Assesses balance between groups based on QIDs. A high value indicates significant differences, suggesting strong QID impact. **Propensity Score Overlap:** Evaluates overlap in propensity scores between groups. Maximized overlap implies limited comparability, while minimized overlap enhances comparability.
- **Causal Importance:** Measures the causal importance of QIDs in a learned causal graph. High causal importance indicates significant relationships with other attributes, marking strong QID candidates.

3.2.2 Data Privacy Metrics

These metrics evaluate the re-identifiability risk associated with QIDs.

- **k-Anonymity:** Quantifies indistinguishability of individuals based on QIDs. Lower values indicate higher re-identification risk.
- **l-Diversity:** Quantifies the diversity of sensitive attributes within QID-defined groups.
- **t-Closeness:** Measures the divergence between sensitive attribute distributions within QID groups and the overall dataset.
- **Generalization Ratio:** Measures the distribution variation.

3.2.3 Data Utility Metrics

These metrics assess the degree to which anonymized datasets retain their analytical utility and predictive power.

- **Mean Squared Error (MSE):** Average squared error in target prediction.
- **Accuracy:** Correct predictions fraction.
- **Range Utility, Distinct Values, Completeness Utility:** Quantify variability, uniqueness, and quality of data within groups.
- **Group Entropy, Information Gain, Gini Index:** Measures related to randomness and impurity within QID groups.
- **Attribute Length Penalty:** A penalty based on the proportion of QIDs.

3.2.4 QID-Specific Metrics

- **Distinction:** Ratio of unique QID values.
- **Separation:** Separability of records based on QIDs.

3.2.5 Performance Metrics

These metrics evaluate the effectiveness of QID detection algorithms against a ground truth.

- **Precision, Recall, F1 Score:** Standard metrics for classification system performance.
- **Jaccard Similarity, Dice Similarity Coefficient:** Measure overlap between predicted and actual QIDs.
- **Specificity, False Positive Rate:** Evaluate true negative rate and proportion of misclassified negatives.
- **F-Beta Score:** Weighted F-score prioritizing precision/recall.

3.3 Optimization Algorithms

`QIDLearningLib` includes various optimization algorithms to select the most appropriate set of QIDs based on the defined metrics.

- **Evolutionary Algorithm (EA):** A metaheuristic inspired by natural selection, suitable for complex, high-dimensional search spaces where multiple conflicting objectives (privacy vs. utility) must be balanced. It explores broadly and can escape local optima but may require longer runtimes. Time complexity is approximately $O(G \times P \times E)$, where G = generations, P = population size, and E = evaluation cost per individual.
- **Simulated Annealing (SA):** A probabilistic iterative method that balances exploration and exploitation via a temperature parameter. Best used when the search space is moderately large and the risk of local optima is significant. It generally has lower complexity than EA, about $O(I \times E)$, with I iterations.
- **Greedy Search:** A fast, deterministic method suitable for small or well-structured problems where local optima are not problematic. It evaluates the immediate neighborhood and moves only to better solutions, thus it may get stuck in local optima. Its time complexity is roughly $O(I \times N \times E)$, where I is iterations and N is neighborhood size.

Algorithm selection guidelines:

- Use **EA** for complex datasets with many attributes and when balancing multiple privacy and utility metrics.
- Use **SA** when moderate search complexity is needed and some local optima avoidance is required with limited runtime.
- Use **Greedy Search** for quick, approximate results on smaller datasets or when computation time is highly constrained.

4 Utilizing `QIDLearningLib` for QID Selection

This section details how to use `QIDLearningLib` for QID selection, focusing on the provided Evolutionary Algorithm (EA) example and outlining a practical pipeline for integration with anonymization tools.

Practical Workflow:

1. **Load dataset:** Import the target dataset into `QIDLearningLib` in a supported format.
2. **Configure algorithm:** Choose the selection algorithm (e.g., EA, Simulated Annealing, Greedy Search) and set the privacy-utility trade-off metrics according to your needs.

3. **Run QID selection:** Execute the optimization process to identify the optimal set of quasi-identifiers.

4. **Export results:** Save the selected QID combinations and their evaluated metric values to a CSV file for easy sharing and downstream use.

5. **Import to anonymizers:** Load the exported QID list into anonymization frameworks such as ARX [**?**] or Amnesia [**?**]. Note that these tools currently require manual QID specification.

6. **Perform anonymization:** Use the specified QIDs to anonymize the dataset, balancing privacy and utility as guided by the selected metrics.

This pipeline enables a streamlined, reproducible workflow that bridges automated QID discovery and practical anonymization, improving efficiency and reducing manual trial-and-error.

4.1 Loading Dataset and Defining Ground Truth

First, you need to load your tabular dataset and define your ground truth QIDs if available. The ground truth QIDs are used for evaluating the performance of the QID selection system.

4.2 Defining Metrics

The core of QID selection in `QIDLearningLib` involves defining the metrics that the optimization algorithm will use. You instantiate `Metric` objects, specifying their name, weight, the metric function, and whether to maximize or minimize the metric.

4.3 Configuring and Executing the Evolutionary Algorithm

The Evolutionary Algorithm is a powerful tool for finding an optimal set of QIDs. You need to configure its parameters such as population size, number of generations, crossover and mutation rates, and elite size.

4.4 Retrieving Selected QIDs and Evaluation

After the EA runs, you can retrieve the selected QIDs and compute various evaluation metrics to assess the performance of the selection.

4.5 Code Example: Evolutionary Algorithm for QID Selection

The following Python code demonstrates how to use the Evolutionary Algorithm for QID selection and evaluate its performance.

```python
import numpy as np
import pandas as pd
from QIDLearningLib.optimizer.metric import Metric
from QIDLearningLib.optimizer.ea import EvolutionaryAlgorithm
from QIDLearningLib.metrics.performance import specificity, recall, accuracy

# Load dataset
file_path = "datasets/sample_dataset.csv"
df = pd.read_csv(file_path)

# Define ground truth QIDs
ground_truth_qids = {"Age", "Gender", "Zipcode"} # Example predefined QIDs

# Define metrics
metrics = [
    Metric("Distinction", 5, Metric.distinction, maximize=True),
    Metric("Separation", 0.5, Metric.separation, maximize=True),
    Metric("k-Anonymity", -0.4, Metric.k_anonymity, maximize=True),
    Metric("Delta Distinction", 0.2, Metric.delta_distinction, maximize=True),
    Metric("Delta Separation", 0.2, Metric.delta_separation, maximize=True),
    Metric("Attribute Length Penalty", -1, Metric.attribute_length_penalty, maximize=True)
]

# Configure and execute the Evolutionary Algorithm
ea = EvolutionaryAlgorithm(df, metrics, alpha=5, population_size=50, generations=30,
                initial_crossover_rate=0.3, initial_mutation_rate=0.2,
                elite_size=1, tournament_size=5, interactive_plot=True)

best_individual, best_fitness, history = ea.run()

# Retrieve selected QIDs
selected_indices = np.where(best_individual == 1)[0]
selected_attributes = set(df.columns[selected_indices])

# Compute evaluation metrics
attributes_set = set(df.columns) # Universal set of attributes
spec = specificity(selected_attributes, ground_truth_qids, attributes_set)
rec = recall(selected_attributes, ground_truth_qids)
acc = accuracy(selected_attributes, ground_truth_qids)

# Optionally, plot metric evolution
# plot_evolution(history, metrics, ea.generations)
```

Listing 1: Python code to run the QID-selecting EA on a dataset and evaluate performance

4.5.1 Code Explanation and Guidelines

- `import numpy as np`, `import pandas as pd`: Standard libraries for numerical operations and data manipulation.
- `from QIDLearningLib.optimizer.metric import Metric`: Imports the `Metric` class for defining custom metrics.
- `from QIDLearningLib.optimizer.ea import EvolutionaryAlgorithm`: Imports the `EvolutionaryAlgorithm` class, which is the optimizer used for QID selection.
- `from QIDLearningLib.metrics.performance import specificity, recall, accuracy`: Imports performance metrics to evaluate the selected QIDs.
- `file_path = "datasets/sample_dataset.csv"`: Replace this with the actual path to your tabular dataset. The dataset should be in a format readable by pandas (e.g., CSV, Excel).
- `df = pd.read_csv(file_path)`: Loads the dataset into a pandas DataFrame.
- `ground_truth_qids = {"Age", "Gender", "Zipcode"}`: This is a crucial step if you have prior knowledge of true QIDs. Define a set of these attributes. If you don't have ground truth, you can skip the performance evaluation part or adapt it to evaluate against a different benchmark.
- `metrics = [...]`: This list defines the optimization objectives. Each `Metric` object takes:
  - **Name (string):** A descriptive name for the metric (e.g., "Distinction").
  - **Weight (float):** The importance of this metric in the overall fitness calculation. Positive weights for maximization, negative for minimization.
  - **Metric Function (`Metric.<metric_name>`):** The actual function from `QIDLearningLib.optimizer.metric` (or other modules) that calculates the metric.
  - **`maximize=True/False`:** A boolean indicating whether the objective is to maximize or minimize this metric. For example, 'Distinction' and 'Separation' are typically maximized, while 'k-Anonymity' (when framed as re-identification risk) and 'Attribute Length Penalty' might be minimized (hence the negative weight and maximize=True effectively minimizing the negative value).
- `ea = EvolutionaryAlgorithm(...)`: Initializes the EA. Key parameters include:
  - `df`: The DataFrame containing your data.
  - `metrics`: The list of `Metric` objects defined previously.

  - `alpha`: A parameter for the fitness function, usually influencing the balance between different metrics.
  - `population_size`: Number of individuals in each generation of the EA. Larger populations can lead to better solutions but increase computation time.
  - `generations`: Number of iterations the EA will run. More generations can lead to convergence but also increase computation time.
  - `initial_crossover_rate`, `initial_mutation_rate`: Probabilities for genetic operations (crossover and mutation). These control the exploration-exploitation trade-off.
  - `elite_size`: Number of top individuals to carry over to the next generation without modification (elitism).
  - `tournament_size`: Size of the tournament selection used to choose individuals for reproduction.
  - `interactive_plot=True`: If set to True, the EA might display an interactive plot of metric evolution during the run (requires appropriate plotting backends).

- `best_individual, best_fitness, history = ea.run()`: Executes the EA.
  - `best_individual`: A binary vector representing the best set of QIDs found (1 if selected, 0 if not).
  - `best_fitness`: The fitness score of the best individual.
  - `history`: A record of metric evolution over generations, useful for analysis and plotting.

- `selected_indices = np.where(best_individual == 1)[0]`: Converts the binary representation of the best individual into indices of selected attributes.

- `selected_attributes = set(df.columns[selected_indices])`: Gets the names of the selected QIDs.

- `spec = specificity(...), rec = recall(...), acc = accuracy(...)`: Calculates performance metrics by comparing the `selected_attributes` against the `ground_truth_qids`. `attributes_set` is the universal set of all attributes in your dataset.

- `plot_evolution(history, metrics, ea.generations)`: (Commented out in the example) This function, if implemented and available, can visualize how the metrics changed over the generations, helping to understand the EA's convergence.

4.6 Tips for Effective Utilization

- **Dataset Preparation:** Ensure your tabular dataset is clean and preprocessed. Handle missing values, categorical features, and outliers appropriately before feeding it to `QIDLearningLib`.
- **Ground Truth QIDs:** If possible, define a ground truth set of QIDs. This allows for objective evaluation of the selection algorithm's performance using metrics like specificity, recall, and accuracy.
- **Metric Selection and Weighting:**
  - Carefully choose the metrics relevant to your privacy and utility goals. The library offers a wide range of metrics, and not all may be applicable or equally important for every scenario.
  - Adjust the weights of the metrics to reflect their importance. For instance, if privacy is paramount, you might give higher negative weights to privacy risk metrics.
  - Consider the interdependencies between metrics. The paper mentions that causality, data privacy, and data utility are not independent, and choosing metrics across different domains or with lower correlation can reduce redundancy and ensure a more complete evaluation.
- **Optimization Algorithm Parameters:**
  - **Population Size and Generations:** These parameters significantly impact the EA's exploration and convergence. Larger values increase the chance of finding better solutions but require more computational resources and time. Start with moderate values and adjust as needed.
  - **Crossover and Mutation Rates:** These genetic operators control the diversity of the population and the ability to escape local optima. Experiment with different values (e.g., 0.6-0.9 for crossover, 0.01-0.1 for mutation).
  - **Alpha:** This parameter often balances the influence of different metrics within the fitness function. Fine-tune it to achieve the desired trade-off between privacy and utility.
- **Interpreting Results:**
  - The `best_individual` represents the selected QID set. Analyze these attributes in the context of your dataset.
  - The `best_fitness` value indicates how well the selected QID set performs against your defined metrics and their weights.
  - Performance metrics (specificity, recall, accuracy) provide a quantitative assessment of how accurately the algorithm identified the ground truth QIDs.

  - If available, use the interactive plots or visualize the `history` to observe the evolution of metrics and fitness over generations. This can provide insights into the convergence behavior of the EA.

- **Experimentation:** QID selection is often an iterative process. Experiment with different metric combinations, weights, and EA parameters to find the optimal configuration for your specific dataset and privacy requirements.
- **Other Optimization Algorithms:** While the example focuses on the Evolutionary Algorithm, remember that `QIDLearningLib` also supports Simulated Annealing and Greedy Search. You might explore these algorithms for comparison or if they better suit your problem's characteristics.

5 Exporting Results

After running the QID selection process and evaluating the performance, you might want to export the results for further analysis, reporting, or sharing. You can easily export the selected QIDs and the computed evaluation metrics to a CSV file using standard Python libraries like Pandas.

```
# Assuming 'selected_attributes', 'spec', 'rec', 'acc' are already computed
results_data = {
    "Metric": ["Selected QIDs", "Specificity", "Recall", "Accuracy"],
    "Value": [
        ", ".join(list(selected_attributes)), # Convert set to list and then join for CSV
        spec,
        rec,
        acc
    ]
}

results_df = pd.DataFrame(results_data)
results_df.to_csv("qid_selection_results.csv", index=False)
print("Results exported to qid_selection_results.csv")
```

Listing 2: Python code to export QID selection results to CSV

This code snippet creates a Pandas DataFrame from the computed results and then saves it as a CSV file named `qid_selection_results.csv`. The `index=False` argument prevents Pandas from writing the DataFrame index as a column in the CSV.

6 Metric Correlation Analysis

Understanding the correlations between different metrics is crucial for effective QID selection. It helps in identifying redundant metrics, understanding trade-offs, and selecting a diverse set of metrics for optimization. The

`QIDLearningLib` offers specific functions to calculate various metrics, which can then be used to compute their correlations across different quasi-identifier combinations.

6.1 Implementing Metric Calculation and Correlation

To calculate the correlation matrix, you first need to define the individual metric functions and then a helper function that computes the correlation across a range of QID combinations.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from itertools import combinations
from QIDLearningLib.metrics.privacy import k_anonymity
from QIDLearningLib.metrics.utility import group_entropy,
    gini_index, information_gain, range_utility, mean_squared_err
from QIDLearningLib.metrics.qid import distinction, separation
from QIDLearningLib.metrics.causality import causal_importance #
    Assuming this exists or is a placeholder

# Load dataset (assuming df is already loaded as in previous
    examples)
# file_path = "datasets/sample_dataset.csv"
# df = pd.read_csv(file_path)

sensitive_attribute = 'Disease' # Example sensitive attribute
true_values = df['Income'] # Example true values for utility
    metrics
columns_excluding_id = [col for col col in df.columns.tolist() if
    col != 'ID']

# Example quasi-identifier combinations (e.g., all combinations of
    3 attributes)
# For a real dataset, consider a subset or a more targeted
    selection
quasi_identifiers_combinations = list(combinations(
    columns_excluding_id, 3))

# Define the metrics you want to compare
# Each lambda function should return a single scalar value for a
    given combo
metrics_to_correlate = {
    "Entropy": lambda data_frame, combo: group_entropy(data_frame,
    list(combo)).mean(),
    "Gini Index": lambda data_frame, combo: gini_index(data_frame,
    list(combo), 'Income').mean(),
    "Information Gain": lambda data_frame, combo: information_gain(
    data_frame, list(combo), 'Income').mean(),
    "Range Utility": lambda data_frame, combo: range_utility(
    data_frame, list(combo), 'Income').mean(),
    "MSE": lambda data_frame, combo: mean_squared_err(data_frame,
    list(combo), 'Income', true_values).mean(),
    "k-anonymity": lambda data_frame, combo: k_anonymity(data_frame
    , list(combo)).mean(),
```

```
    "Causal Importance": lambda data_frame, combo: causal_importance(data_frame, list(combo)), # Placeholder/example
    "Distinction": lambda data_frame, combo: distinction(data_frame, list(combo)),
    "Separation": lambda data_frame, combo: separation(data_frame, list(combo)),
}

def calculate_metric_correlations(df, metrics_dict, qid_combinations):
    """
    Calculates the correlation matrix between different metrics across QID combinations.

    Args:
        df (pd.DataFrame): The input dataset.
        metrics_dict (dict): A dictionary of metric names and their corresponding lambda functions.
        qid_combinations (list): A list of quasi-identifier combinations (tuples of column names).

    Returns:
        pd.DataFrame: A correlation matrix of the metrics.
    """
    metric_results = {name: [] for name in metrics_dict.keys()}

    for combo in qid_combinations:
        # Ensure combo is a list for functions expecting it
        current_combo_list = list(combo)
        for metric_name, metric_func in metrics_dict.items():
            try:
                # Pass the DataFrame and the current combination to the metric function
                value = metric_func(df, current_combo_list)
                metric_results[metric_name].append(value)
            except Exception as e:
                # Handle cases where a metric might not be calculable for a combo
                # print(f"Warning: Could not calculate {metric_name} for {combo}: {e}")
                metric_results[metric_name].append(np.nan) # Append NaN for missing values

    metrics_df = pd.DataFrame(metric_results)
    return metrics_df.corr()

# Calculate the correlation between different metrics
correlation_matrix = calculate_metric_correlations(df, metrics_to_correlate, quasi_identifiers_combinations)

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", vmin=-1, vmax=1, fmt=".2f")
plt.title('Correlation Between Different Metrics Across QID Combinations')
```

```
plt.show()

print("\nMetric Correlation Matrix:")
print(correlation_matrix)
```

Listing 3: Python code for detailed Metric Correlation Analysis

6.2 Code Explanation and Guidelines for Correlation Analysis

- `from itertools import combinations`: This import is essential for generating all possible combinations of attributes to be considered as QIDs.
- The present imports bring in the specific metric functions from `QIDLearningLib` that you wish to analyze for correlation. Ensure the paths and function names match your library's structure.
- `sensitive_attribute = 'Disease'`, `true_values = df['Income']`: Define your sensitive attribute and true values (e.g., for target variable in utility metrics) according to your dataset.
- `columns_excluding_id = [col for col in df.columns.tolist() if col != 'ID']`: This creates a list of potential QID columns, excluding any explicit 'ID' columns.
- `quasi_identifiers_combinations = list(combinations(columns_excluding_id, 3))`: This generates all unique combinations of 3 attributes from your potential QID columns. You can change the number '3' to explore combinations of different sizes. Be mindful that increasing this number significantly increases computation time due to the exponential growth of combinations.
- `metrics_to_correlate = {...}`: A dictionary where keys are the names of the metrics (e.g., "Entropy", "k-anonymity") and values are lambda functions. Each lambda function takes the DataFrame (`data_frame`) and a QID combination (`combo`) as input, and should return a single numerical value for that metric. Note the use of `.mean()` for metrics that might return a Series of values (e.g., group-wise metrics).
- `calculate_metric_correlations(df, metrics_dict, qid_combinations)`: This custom function iterates through each defined QID combination, calculates the value for each specified metric, and stores these values. It then forms a DataFrame from these results and computes the correlation matrix using `metrics_df.corr()`.
- `sns.heatmap(...)`, `plt.title(...)`, `plt.show()`: These lines generate a heatmap of the correlation matrix. The heatmap visually represents the pairwise correlation coefficients between each pair of metrics.

- **Interpretation:**
  - Values close to +1 indicate a strong positive linear correlation (metrics tend to increase or decrease together).
  - Values close to -1 indicate a strong negative linear correlation (as one metric increases, the other tends to decrease).
  - Values close to 0 indicate a weak or no linear correlation.

Guidelines for Metric Selection based on Correlation:

- **Reduce Redundancy:** If two metrics show a very high positive or negative correlation (e.g., > 0.8 or < -0.8), they might be measuring very similar aspects. You might consider choosing only one of them to reduce the complexity of your optimization problem and ensure that each metric contributes unique information.
- **Ensure Diversity:** Aim for a set of metrics that are not highly correlated. This ensures that your QID selection process considers various aspects of privacy, utility, and causality, leading to a more robust and balanced solution.
- **Understand Trade-offs:** Often, privacy metrics might be negatively correlated with utility metrics. Understanding these trade-offs through the correlation matrix is crucial for making informed decisions during QID selection and setting appropriate weights for your optimization objectives.

7 Conclusion

`QIDLearningLib` offers a robust and flexible framework for automated QID recognition and evaluation in tabular datasets. By understanding its core concepts, metrics, and optimization algorithms, users can effectively manage privacy risks while preserving data utility. The ability to export results and perform detailed correlation analysis further enhances the utility of the library for comprehensive data privacy research. Continuous experimentation and fine-tuning of parameters are key to achieving optimal results for diverse datasets and privacy objectives.

8 References

- The original paper: Sancho Amaral Simões, João P. Vilela, Miriam Seoane Santos, Pedro Henriques Abreu. "QIDLEARNINGLIB: A Python Library for Quasi-Identifier Recognition and Evaluation". Preprint submitted to Elsevier.