Detailed Code Development Documentation

Title: Development of an Entity Resolution System to Identifya and group duplicate records in a Company Dataset
Author: Francesco Grasso
Date: March 14, 2025
Context: The dataset contains company records imported from multiple systems, leading to duplicate entries with slight variations.

Table of contents

Introduction

Entity resolution is the process of identifying and linking records that refer to the same entity (e.g., companies with slight variations in names or addresses). This project aims to:

- **Preprocess the data**: Normalize names, URLs, and phone numbers to standardize formats.
- **Optimize blocking**: Group similar records to reduce comparisons.
- **Train a model**: Use supervised machine learning to classify duplicates.
- **Deduplicate records**: Produce a clean dataset with unified entities.

I chose the **Dedupe** library for its flexibility in handling heterogeneous data, ability to learn from supervised training data, and interactive labeling interface. This project reflects an iterative approach, with progressive optimizations to balance performance and results.

Code Evolution

The development occurred in several stages, each addressing specific technical challenges:

1. Initial Version

- **Objective**: Create a basic working system.
- **Implementation**:
    - Loaded the dataset (e.g., CSV with columns like name, URL, address, phone, country).
    - Preprocessing: Normalized data (e.g., removed legal suffixes from names).
    - Blocking: Created keys based on name, URL, address, and country.
    - Interactive training: Used console_label to manually label record pairs as duplicates or distinct, saving results in training.json.
    - Deduplication: Clustered similar records and saved the output.
- **Result**: A functional prototype but inefficient for large datasets.

2. Blocking Optimization

- **Issue**: Analysis showed too many small blocks (average of 1.93 records per block), with many containing only one record, increasing unnecessary comparisons.
- **Solution**:
    - Modified the create_blocking_key function to use only name and country, reducing the number of keys and increasing average block size.
- **Rationale**: Reduce computational load while maintaining meaningful groupings.

3. Memory Management

- **Issue**: Processing large datasets caused memory issues.
- **Solution**:
    - Introduced process_block_batch: Processed blocks in batches rather than all at once.
    - Used gc.collect() to free memory between batches.
- **Rationale**: Ensure scalability on real-world datasets, avoiding hardware crashes.

4. Separation of Interactive Labeling

- **Objective**: Improve modularity and reuse of training data.
- **Implementation**:
    - Created a separate script (interactive_labeling.py) for interactive labeling, saving results in training_data.json.
    - Modified the main code to load the pre-existing training file and proceed with deduplication.
- **Rationale**: Separate training and deduplication phases for flexibility and easier debugging.

5. Debugging and Robustness

- **Objective**: Resolve errors and improve stability.
- **Implementation**:
    - Added checks to ensure required fields were present in records.
    - Introduced logging to track execution and identify issues.
- **Rationale**: Make the system robust and capable of handling incomplete or malformed data.

Errors Encountered and Solutions

During development, I faced several technical challenges, resolving them with targeted solutions:

1. Memory Error

- **Issue**: MemoryError when processing large blocks.

- **Solution**: Implemented process_block_batch to work on small groups of records and freed memory with gc.collect().
- **Rationale**: Prevent system crashes on large datasets, maintaining scalability.

2. Data Validation Error

- **Issue**: ValueError: Records do not line up with data model when a field (e.g., clean_company_name) was missing in a record.
- **Solution**: Added checks to set missing fields to None and ensure all records conformed to the model.
- **Rationale**: Make the code resilient to imperfect data, typical in real-world datasets.

3. Inefficient Blocks

- **Issue**: Too many single-record blocks slowed the process without improving results.
- **Solution**: Simplified blocking keys (only name and country), creating larger, more useful blocks.
- **Rationale**: Optimize performance by reducing the total number of comparisons.

4. Invalid Training Data

- **Issue**: Errors loading training_data.json due to malformed records.
- **Solution**: Rigorous cleaning of training data, checking for fields and validity.
- **Rationale**: Ensure the model learned from consistent data, improving accuracy.

Architectural and Technological Choices

The decisions reflect a balance between accuracy, scalability, and simplicity:

1. Library Choice: Dedupe

- **Why Dedupe?**:
    - Flexibility in handling heterogeneous data (names, addresses, URLs, phones).
    - Support for supervised training via interactive labeling.
    - Built-in clustering algorithms for deduplication.
- **Alternative rejected**: Custom algorithms (e.g., Levenshtein) would require more time and code without Dedupe's scalability.

2. Data Preprocessing

- **Company Names**: clean_company_name function to remove legal suffixes (e.g., "LLC", "Inc") and irrelevant characters.
    - **Rationale**: Standardize names for better comparison.
- **URLs**: Normalization with normalize_url to standardize format (e.g., removing "www").
    - **Rationale**: Reduce irrelevant variations.

- **Phones**: Conversion to E164 format using the phonenumbers library.
    - **Rationale**: Ensure consistency in international numbers.

3. Blocking Strategy

- **Approach**: Used doublemetaphone for names (phonetic) and parts of the URL as keys.
- **Optimization**: Reduced attributes to create larger blocks.
- **Rationale**: Balance efficiency (fewer blocks) and accuracy (meaningful groupings).

4. Memory Management

- **Technique**: Batch processing with process_block_batch and gc.collect().
- **Rationale**: Adapt the system to varying dataset sizes, avoiding overloads.

5. Code Modularity

- **Choice**: Separated interactive labeling into a dedicated script.
- **Rationale**: Facilitate reuse of training data and simplify debugging, making the code more maintainable.