User Manual for **AI_VLookUp** Fuzzy Matching Program

Luis Lascano

March 17, 2025

Contents

1 Introduction

This user manual describes how to utilize the **AI_VLookUp Fuzzy Matching Program**, a Java-based solution that performs fuzzy matching across two Excel datasets (a "reference" dataset and a "messy" or incomplete dataset). The program relies on a YAML configuration file to define how the columns (or groups of columns) between these two datasets should be mapped and compared.

1.1 Scope

The manual focuses on:

- Installing and running the **AI_VLookUp** fuzzy matcher.
- Understanding and configuring the YAML file.
- The structure of the input Excel files.
- Best practices for ensuring accurate and efficient matching.

1.2 High-level Overview

The core functionality of **AI_VLookUp** is to match entries (rows) from one table (often called the "messy" or incomplete table) to entries in a second reference table. The matching leverages:

- Tokenization of cell contents.
- Fuzzy-matching techniques (Jaccard similarity, Damerau-Levenshtein distance, etc.).
- Weighted groupings of columns, controlled by a *softmax* function applied to numeric weights.

2 Basic Usage

This section explains what you need to run the program, how to build it, and how to execute it with minimal fuss.

2.1 System Requirements

- **Java** (version 1.8 or newer).
- **Maven** (if you wish to build from source).
- **Excel files in `.xlsx` format** for your reference and messy tables.
- A suitable environment (Windows, Linux, or macOS).

2.2 Dependencies and Installation

1. Verify your Java version:

   ```
   java -version
   ```

2. Install Maven (example for macOS using Homebrew):

   ```
   brew install maven
   ```

3. Clone or download the repository:

   ```
   git clone https://github.com/luislascano01/AI_VLookUp
   cd AI_VLookUp
   ```

4. Build the program (JAR) from source:

   ```
   cd Engine
   mvn clean install
   ```

5. A JAR file (e.g., `ai`$_v lookup-0.0.1-SNAPSHOT.jar)willbecreatedinthe$`Engine/target`$directory$.

2.3 Running the Program

Assuming you have your YAML configuration file ready (see Section 4 for details on creating or editing it), you can run:

```
java -jar Engine/target/ai_vlookup-0.0.1-SNAPSHOT.jar path/to/your_config.yaml
```

For the sample configuration provided in the project:

```
java -jar Engine/target/ai_vlookup-0.0.1-SNAPSHOT.jar \
    Engine/src/main/resources/header_configuration.yaml
```

The results are generated in your specified `OperatingDir` (see Section 6), typically as `results.csv`.

3 Excel File Requirements

The program automatically processes all rows in the specified sheet. However, consider:

- **Sheet Names:**
  - The **Reference Table** must be on a sheet named `Primary_Reference_Table`.
  - The **Messy Table** must be on a sheet named `Secondary_Data`.
- **File Format:** Files must be in `.xlsx` format (no `.xls`).
- **Headers:**
  - Headers should ideally use underscores (_) instead of spaces. Spaces in headers are not fully tested and may cause parsing issues.
  - There should be no extra placeholders or counters below the last data row.
- **Static Data:** Cells should contain static text/number data rather than formulas. If formulas exist, their resulting values may not be properly interpreted.

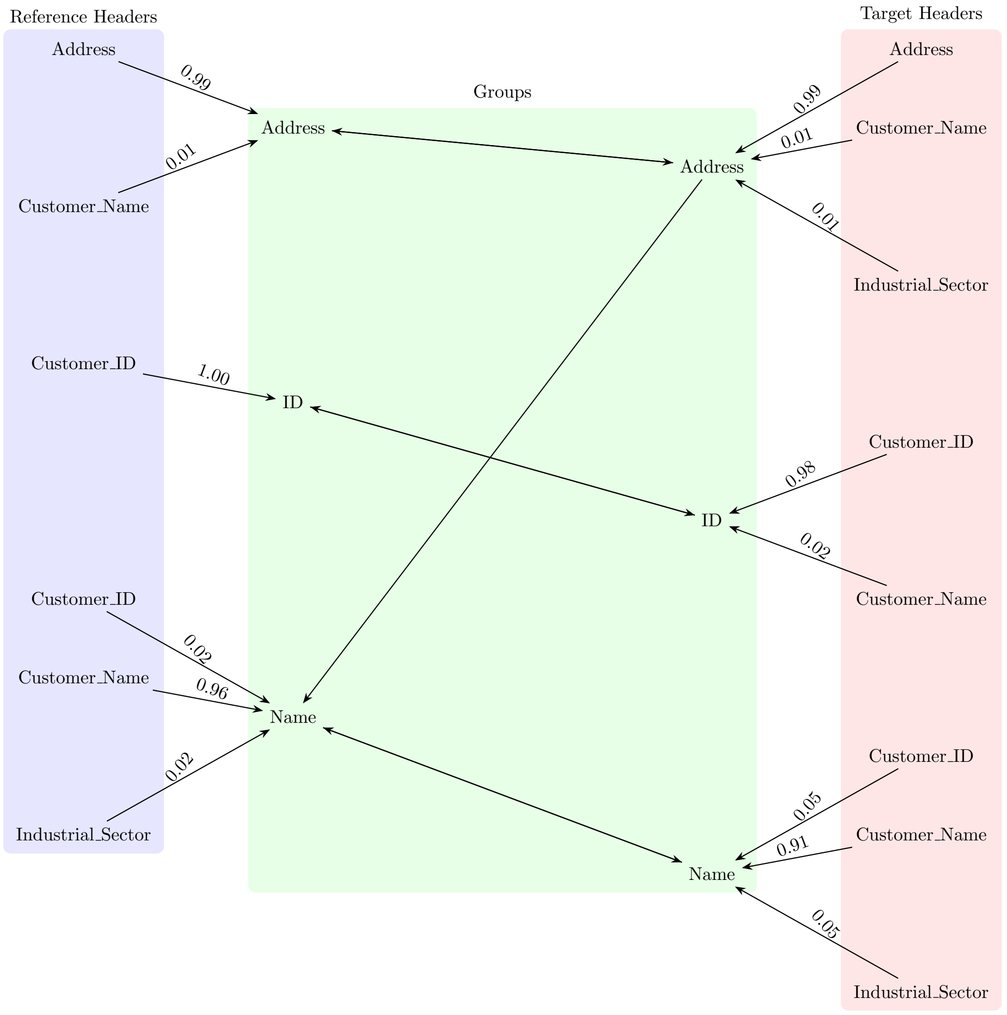
4 YAML Configuration File

The YAML file is essential for configuring how columns in the reference ("clean") table align with columns in the messy ("incomplete") table and the weight or importance of those columns.

4.1 Sample configuration

```yaml
FuzzyDatabaseConfig:
  BackboneConfiguration:
    reference_groups:
      ID:
        - "Customer_ID(1)"   # Must be at least one column in each group
      Name:
        - "Customer_ID(2)"
        - "Customer_Name(6)"
        - "Industrial_Sector(2)"
      Address:
        - "Customer_Name(1)"
        - "Address(6)"

    target_groups:
      ID:
        - "Customer_ID(5)"
        - "Customer_Name(1)"
      Name:
        - "Customer_Name(4)"
        - "Customer_ID(1)"
        - "Industrial_Sector(1)"
      Address:
        - "Customer_Name(1)"
        - "Address(6)"
        - "Industrial_Sector(1)"

    ref_to_tgt:
      ID: "ID"
      Name: "Name"
      Address: "Address"

    tgt_to_ref:
      ID: "ID"
      Name: "Name"
      Address:
        - "Address"
        - "Name"

    reference_key_col: "Customer_ID"
  target_key_col: "Customer_ID"

DataToConsume:
  ReferenceTable: "./Sample_Dataset/Primary_Reference_Table.xlsx"
  MessyTable: "./Sample_Dataset/Secondary_Data.xlsx"

OperatingDir: "./OperatingDir"

RegexPreprocessing:
  Customer_ID: "((?<![/\\d])\\d+(?![/\\d]))"
```

4.2 Sample mapping visualization



4.3 Mandatory vs. Optional Sections

- **BackboneConfiguration** is mandatory. It dictates how columns (grouped by `ID`, `Name`, `Address`, or any *customly named* group) are weighted and mapped. You can freely rename or add additional groups to match your data.

- Inside each group under `reference_groups` or `target_groups`, **at least one column must be listed**. Each column name must match exactly what appears in your Excel headers (underscored if possible).
- The `reference_key_col` and `target_key_col` fields are **mandatory**, but they do not necessarily have to be columns labeled "ID." They can be any stable unique column in each table.
- `DataToConsume` contains:
  - `ReferenceTable` (Excel file path, **mandatory**)
  - `MessyTable` (Excel file path, **mandatory**)
- `OperatingDir` is **mandatory**. The program will write results to this directory; it must be read/write accessible.
- `RegexPreprocessing` is **optional**. You can leave it blank or specify one regex per column. Only a single regex can be mapped to a given column (e.g., `Customer_ID` in the example).

4.4 BackboneConfiguration Details

reference_groups and **target_groups** define how columns within the reference and messy tables are logically grouped for fuzzy matching:

- Each entry looks like `"HeaderName(weight)"`, where `HeaderName` is an exact column header from Excel, and `(weight)` is an integer reflecting that column's importance.
- A higher integer weight produces an *exponentially greater* influence on the softmax weighting used in the matching algorithm. Even moderate numeric differences in weights can lead to significantly different match priorities.
- Group names (such as `ID`, `Name`, or `Address`) are arbitrary; you can label groups however you wish. The critical point is that each group collectively represents columns meant to be compared or fused together.
- If a cell value in a row is empty for a particular column, that row is **not skipped**. The matching algorithm will simply use the available data from other columns in that group and continue.

4.5 Mappings: `ref_to_tgt` and `tgt_to_ref`

- `ref_to_tgt` indicates how a particular *group* in the reference table aligns to a group in the target (messy) table for forward comparison.
- `tgt_to_ref` performs the same mapping in reverse, allowing for a many-to-many relationship between reference and target groups.
- For instance, `ID: "ID"` means the group labeled `ID` in the reference is mapped to the group labeled `ID` in the target. If we see `Address: ["Address", "Name"]` under `tgt_to_ref`, it means that from the target's perspective, the `Address` group should be compared to both `Address` and `Name` in the reference.
- While you can specify these mappings in a single line, it is recommended to use a **multiline** list notation (as in the example) to maintain clarity and avoid errors.

5 Advanced Details

5.1 Softmax Weighting

Although the concept of *softmax* might seem technical, the user only needs to know that **higher numeric values inside the parentheses reflect higher relative importance in matching.** This importance is not linear; instead, columns with higher weights will more strongly influence the overall match scores. For example, a column assigned `(6)` has a much greater *weight* than a column assigned `(1)`.

5.2 Tokenization and Fuzzy Metrics

Internally, the program:

1. **Tokenizes** the contents of each cell (breaking text into smaller units).
2. **Assigns a weight** to each token based on:
   - The *softmax* weight of the column in which it appears.
   - Characteristics of the token (e.g., token length, presence of numeric content).
3. Computes **similarity** between tokens across reference and target groups using advanced metrics (like Jaccard or Damerau-Levenshtein).

While this process is technical, it is automatic and requires no additional configuration beyond specifying your desired columns and their relative weights.

6 Operating Directory and Output

The `OperatingDir` path in the YAML configuration must be a valid folder where the program can write outputs. After execution, a typical output file `results.csv` will appear in this folder. You may configure alternative output filenames or formats if your version of the software supports it.

7 Sample Workflow

7.1 Step-by-step Example

1. **Prepare two Excel files**:
   - `Primary_Reference_Table.xlsx` with a sheet named `Primary_Reference_Table`.
   - `Secondary_Data.xlsx` with a sheet named `Secondary_Data`.

   Ensure your column headers use underscores instead of spaces (`Customer_Name`, `Industrial_Sector`, etc.).
2. **Clone or obtain** the project code (if not already done):

```
git clone https://github.com/luislascano01/AI_VLookUp
```

3. **Build the project** via Maven:

```
mvn clean install
```

4. **Edit the YAML configuration** (e.g., `header_configuration.yaml`) to match your file paths, groups, and column headers.
5. **Run the program:**

```
java -jar Engine/target/ai_vlookup-0.0.1-SNAPSHOT.jar \
   Engine/src/main/resources/header_configuration.yaml
```

6. **Review output** in the configured operating directory, for example:

```
./OperatingDir/results.csv
```

7.2 Tips and Recommendations

- **Avoid mixing** large numeric columns with purely categorical columns in the same group.
- **Keep your Excel sheets clean:** no extra hidden rows, placeholder text, or formula-driven values if possible.
- **Test small subsets** first to ensure your weights and group mappings produce desired results.

8 Output Interpretation

After running the program, you will typically get a `results.csv` file (or a similar output) in your specified operating directory. A sample snippet from such a results file might look as follows:

query	match	secondMatch	coefficientDamerau	coefficientJaccard	idMatch
0	0	-1	0.859	0.333	1
1	1	-1	0.887	0.375	1
2	2	28	0.164	0.125	0
3	3	29	0.583	0.222	0
4	4	-1	0.475	0.333	1
5	5	38	0.797	0.222	0

Table 1: Example portion of the results table produced by the fuzzy matching program.

- **query:** Refers to the row number (0-based index) in the *messy* table for which you are seeking a match.

- **match:** The row index (0-based) in the *Reference Table* that is suggested as the best match for the **query** row.

- **secondMatch:** A secondary or alternative match candidate from the reference table (could be `-1` if there is no second match).
- **coefficientDamerau** & **coefficientJaccard:** These are similarity coefficients (based on Damerau-Levenshtein and Jaccard algorithms) computed for **additional insights** or manual review. They are not the final determinants for fuzzy matching but can help you spot-check borderline cases.
- **idMatch:** A binary indicator (0 or 1) showing whether the match was considered a valid ID-based match (for instance, if the program recognized that the ID itself is identical or close enough to confirm).

Note that these coefficients may assist users in *further manual cleaning* if needed. They are not necessarily the primary drivers of the fuzzy match decision itself, which also considers other tokenization and weighting mechanisms configured in the YAML file.

9 Troubleshooting and FAQs

9.1 Common Issues

- **Program fails with an Excel read error:** Check that your files are indeed in `.xlsx` format, sheet names are correct, and the path in `DataToConsume` is accurate.
- **Output is empty or no matches:** Ensure that your column weights and group mappings (forward vs. backward) are set properly, and that your source data is consistent.
- **Headers with spaces cause parsing failure:** Rename these columns to use underscores.

9.2 Frequently Asked Questions

1. **Q: Can I use formulas inside the Excel cells?**
   A: It is strongly advised to have static data only. Formula-based cells might not parse as expected.
2. **Q: Do I have to use `ID`, `Name`, and `Address` groups?**
   A: No. These are just examples. You can rename them to suit your data, as long as you keep the structure consistent in the YAML and across your files.
3. **Q: How do I adjust column importance for better matching?**
   A: Increase the numeric factor in parentheses (e.g., from 2 to 6) for columns that are more critical in your match process.

10 Conclusion

The **AI_VLookUp Fuzzy Matching Program** offers a flexible and powerful way to align rows across two Excel files, particularly when data is incomplete or inconsistent. The YAML configuration file plays a central role in defining mappings, weighting, and file paths, while the user can simply run the program as a command-line tool to generate the desired matching results. For more information, visit the project repository at `github.com/luislascano01/AI_VLookUp`.

End of User Manual