December 2023

# UDRC MPI Process

Linking individuals across disparate data sets is a core competency of the UDRC. The UDRC matches individuals across data agencies and time. Matching across agency and time presents challenges, namely noisy data. The noise comes in the form of differing variables across agencies, changing PII over time, and potential errors in data entry. In the presence of uncertainty introduced by noise, when matching is solely deterministic, potential matches are lost, decreasing the power of research and potentially biasing results. To address the uncertainty created by noisy data, the UDRC has developed a probabilistic matching system, the Master Person Index (MPI) system. This paper provides a broad technical overview of the fifth version of the UDRC MPI system.

The MPI system is a multi-step technical and statistical solution that handles probabilistic matching over data sets of substantial size, i.e., "big data" (Figure 1). Given the amount of data that the UDRC ingests, the MPI system must find potential matches well and efficiently sort through existing individuals while matching. To efficiently handle millions of rows of data rows, the MPI process's first step is indexing.

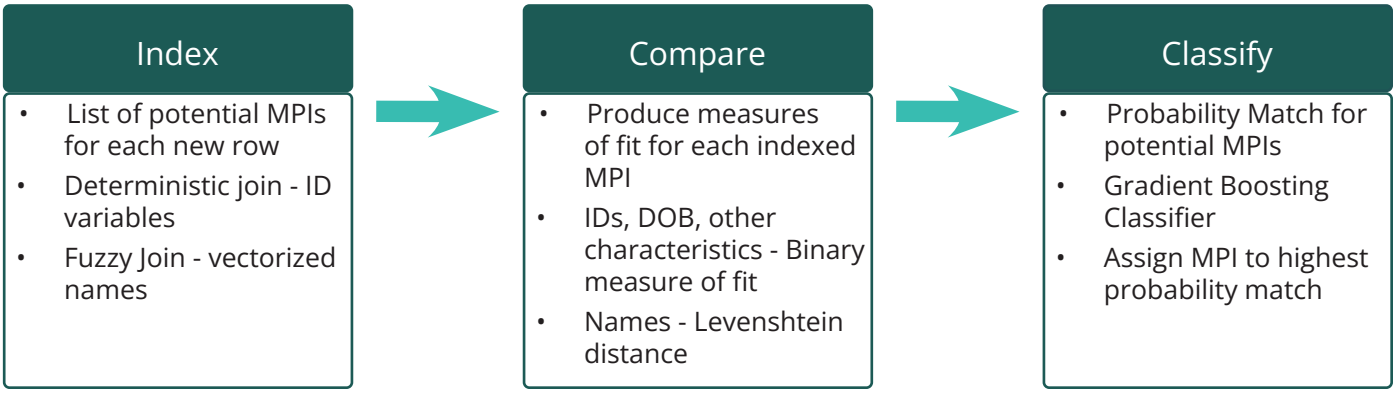


Figure 1: Overview of the MPI Process

Indexing is a two-part process that attaches potential existing identities (MPIs) from the identity pool to the newly added data. There are deterministic and non-deterministic steps to indexing; in the first, potential MPIs are attached to each row of data through a simple join letter procedure iterated over columns that contain IDs. Next, the first and last names are concatenated and then vectorized, converted to a vector of numbers where each number counts the times a appears in a name. This vector is sent through a search tree to efficiently narrow down the potential matches. A search tree narrows down potential matches by testing if a given value falls above or below a specified value; this occurs at a node. This process occurs over multiple nodes, at each node the individual is filtered based on a predetermined vector value to find the nearest neighbors. The search tree was implemented in Python with the Scikit-learn library (Pedregosa et al., 2011). After completion of the search, the MPIs nearest to the vectorized name are added to the index. Table 1 shows an example of index rows. Once the newly added individuals have made it through the indexing process, the next step is to compare potential MPIs.

Table 1: Example of Indexed Rows

| Newly Ingested Row | Indexed MPIs |
|---|---|
| $r_1$ | $[mpi_1, mpi_2, ..., mpi_n]$ |
| $r_2$ | $[mpi_3, mpi_4, ..., mpi_n]$ |

The Compare Process compares the indexed row to the potential MPIs that have been attached to it. This process occurs through a series of BigQuery queries, which methodologically compare the required fields. For columns that contain ID variables, DOB, gender, and race or ethnicity, a binary outcome, match or no match, is assigned. For names, Levenshtein distance (Levenshtein, 1966), the minimum number of single-character edits needed to convert one string to another, is computed and added as the variable. The results are stored in a comparison table and then used to classify each potential MPI.

The Classify Process assigns a probability match for each potential MPI attached to the new individuals. The match probability is determined through a Gradient Boosting Classifier. A gradient-boosting classifier is an ensemble of decision trees designed to minimize a loss function (Natekin & Knoll, 2013); the result is a probability of matching each potential MPI. The potential MPI with the highest probability match is assigned to the individual if it is above a given threshold (Table 2). If there is no MPI above the threshold, this is considered a new individual, and a new MPI is created with that information. The gradient boosting classifier was implemented with Python and the Scikit-learn library (Pedregosa et al., 2011).

Table 2: Example of Assigned Probability Matches

| Newly Ingested Row | Indexed MPIs | Probability Match |
|---|---|---|
| $r_1$ | $mpi_1$ | $P[r_1 = mpi_1]$ |
| $r_1$ | $mpi_2$ | $P[r_1 = mpi_2]$ |
| $r_1$ | $mpi_n$ | $P[r_1 = mpi_n]$ |

The MPI process has been tested on synthetic data that was designed to gauge the accuracy of the MPI system. Synthetic data with increasingly frequent errors were fed into the MPI system. These errors started with a single-letter error in the names and increased to errors in SSN, spelling errors in the names, and changing gender and race or ethnicity. The first tests only included a single error, and the final test included multiple errors for some synthetic individuals. The goal was a 90% or higher probability match for the correct individual. The results (Table 3) show that for each scenario, a minimum of 95% probability match occurred for all individuals that should have matched across data sets, a 100% match rate. These tests set baseline expectations for how the MPI system should perform with real data.

Table 3: MPI Testing Scenarios

| Scenario | Challenge | Expected Minimum Probability Match | Minimum Probability Match | Match Rate |
|---|---|---|---|---|
| 1 | Data Ingestion | 100% | 100% | 100% |
| 2 | Missing Letter in Names | 90% | >95% | 100% |
| 3 | Missing SSID | 90% | >95% | 100% |
| 4 | Change in Gender | 90% | >95% | 100% |
| 5 | Change in Race/Ethnicity | 90% | >95% | 100% |
| 6 | Incorrect DOB | 90% | >95% | 100% |
| 7 | Incorrect SSN | 90% | >95% | 100% |
| 8 | All of the above | 90% | >95% | 100% |

The fifth iteration of the MPI system enables the UDRC to perform its core mission. Through a multi-step process of Indexing, Comparing, and Classifying, the MPI system probabilistically matches individuals in the presence of noisy data. The fifth version of the MPI system handled the synthetic data tests well, matching with greater than 95% probability and a 100% match rate.

# References

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(8), 707–710.

Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. Frontiers in Neurorobotics, 7, 21.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.