Assignment 2: Deduplication and Crawling

Prepared by

Kushagra Gupta

(2025909001)

Under the guidance of

Prof. Anil Nelakanti

Submitted in
partial fulfillment of the requirements
for the course of

CS4.406: Information Retrieval and Extraction

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

HYDERABAD

(November 2025)

Activity 2.1: Deduplication

Goal: To find and group all records belonging to the same person from a noisy dataset of 5,000 entries.

My Approach: Followed a data-cleaning and linkage pipeline:

1. **Exploratory Data Analysis (EDA):** I first inspected the data and found many problems, including missing values, incorrect data types (like date_of_birth as a number), and typos in text fields (like 'nsw' vs. 'nws').
2. **Data Preprocessing:** I cleaned the entire dataset. I fixed all typos, converted dates and postcodes to standard string formats, and filled in missing values.
3. **Blocking:** To avoid comparing 12.5 million pairs, I used a "blocking" strategy. I grouped records by postcode, assuming matches would be in the same postal area.
4. **Pairwise Comparison:** Within these blocks, I used the recordlinkage library to compare pairs. I scored similarity on names, addresses, and date of birth using the Jaro-Winkler method.
5. **Decision & Clustering:** I set a rule that any pair with a total similarity score of 4.0 (out of 5.0) was a "match." I then used the networkx library to find all connected groups, or "clusters."

Key Libraries Used:

- pandas (for data loading and cleaning)
- recordlinkage & jellyfish (for blocking and comparing)
- networkx (for clustering the matched pairs)
- sklearn (for evaluation)

Results & Insights:

- **Insight:** The soc_sec_id column was the "answer key." The 5,000 records only had 2,291 unique soc_sec_ids. I used this to check my work.
- **Result 1 (Blocking**): My blocking strategy was very effective, reducing the number of comparisons by 99.87% (from 12.5 million to just 16,115).
- **Result 2 (Accuracy):** My algorithm found 1,059 clusters. When I compared this to the 2,291 "true" clusters, I got an Adjusted Rand Score (ARS) of 0.7307. This is a great result, showing my model's groups were very similar to the real ones. The lower cluster count means my model tended to "over-cluster," sometimes grouping two different people who just had very similar information.

Activity 2.2: Crawling Summary (v1.2)

Goal To build a bot for the new v1.2 server, which required 15-second submissions within a 60-second window to generate an evaluation.bin file. The bot had to be optimized for a new balance of metrics: coverage, mse, avg_staleness, and visit_count.

Approach: The "Optimal Staleness" Bot I built an asynchronous bot
using asyncio and aiohttp that followed a precise "sleep-then-crawl" strategy.

1. **Initial Crawl:** The bot performed a high-speed BFS crawl, discovering the entire 50+ page graph in ~1.22 seconds.
2. **First Submission:** It immediately calculated PageRank and submitted the first evaluation.
3. **Main Loop:** The bot then entered a loop where it would **sleep** for ~11-13 seconds, then **re-crawl** all 50+ pages *just before* the 14.5s deadline.
4. **Shutdown:** This cycle ensured every submission had fresh node_ids, keeping staleness low. The bot ran until the server returned an "Evaluation window has ended" error (at t=85s), which successfully triggered the evaluation.bingeneration.

Key Libraries Used:

- asyncio & aiohttp (For the high-speed bot)
- beautifulsoup4 (For HTML parsing)
- networkx (For PageRank calculation)

Key Results The "sleep-then-crawl" strategy was critical; a simpler "crawl-then-sleep" approach failed, causing Staleness to explode to over 58,000 ms.

The final valid submission (at t=56.00s) achieved excellent, balanced scores:

- **Coverage:** 96.2% (50/52 pages)
- **MSE:** 1.12e-07 (Near-perfect PageRank accuracy)
- **Staleness:** Remained low and stable (e.g., 1430ms, 6451ms, 10703ms)
- **Visits:** 252
- **Final Outcome:** The bot successfully completed its full lifecycle and generated the evaluation.bin file for submission.

Appendix

GitHub Repo: All the data, code file (ipynb and py), report.

https://github.com/SpyBeast07/HomeworkIRE/tree/main/deduplication_and_crawling