Athens University of Economics and Business

School of Business

Department of Management Science & Technology

Master of Science in Business Analytics

Program:	Full-time
Quarter:	3rd (Spring Quarter)
Course:	Advanced Topics in Data Engineering
Assignment №:	Entity Resolution Assignment
Students (Registration №):	Souflas Eleftherios - Efthymios (f2822217)

Table of Contents

Entity Resolution Assignment

Introduction

This assignment involves working with the provided *ER-Data.csv* file to address several challenges in Entity Resolution. Four distinct tasks are assigned, each contributing to a holistic approach to enhancing data quality and accuracy. Task A focuses on employing the Token Blocking method, a schema-agnostic approach that leverages the attributes of entities to create Key-Value (K-V) pairs. This methodology captures the essence of attributes while excluding the id column (which is going to be used for the association with the token block as value). Task B involves computing the number of comparisons required for resolving duplicates within the created blocks. In Task C, firstly a Meta-Blocking graph is constructed from the block collection generated in Task A, then, the CBS Weighting Scheme (Common Block Scheme) is applied to refine the graph, after that, graph pruning on edges with weight lower than 2 is applied to reduce unnecessary comparisons and finally the number of comparisons after pruning is re-calculated. The final task, Task D, involves designing a function to compute Jaccard similarity based on the 'title' attribute, paving the way for accurate similarity measurement. These tasks collectively form a comprehensive strategy to enhance Entity Resolution accuracy and efficiency.

Task A

A Token Blocking methodology is employed to generate distinct Blocking Keys (BK) represented as keys in the Key-Value (K-V) pairs from *ER-Data.csv*. The primary aim is to enable precise matching and analysis by extracting BKs from entities' attribute values while excluding the id column. The libraries $pandas$ and $html$ were imported, and functions from $nltk$, $pprint$, and $contextlib$ were utilized.

The $token\_blocking(dataframe)$ function constructs an inverted index as a dictionary. For each row in the Data Frame, it extracts IDs and attributes, converts attributes' text to lowercase, splits it into tokens, and filters out English stop-words using NLTK. Tokens that belong to the English list of stop-words, like I, you, of, and, etc., are filtered out. If this step was omitted, the token with the most entities in its set of values would be the token (word) 'of', having 30,326 entities associated to it. After the stop-words filtering step, 'j' is the token with the most values in its block, having 7,599 entities, reducing this way the computation of unnecessary comparisons. Tokens (keys) are associated with entity IDs (values) in the index. The function removes tokens from the index that do not meet the minimum threshold of having at least two entities associated with them (Alexiou, 2023, p. 26).

The $decode\_html\_special\_chars(text)$ function decodes HTML special characters in a given text using $html.unescape$, addressing characters like '&', '<', '>', etc.

The *ER-Data.csv* file is firstly loaded into a $pandas$ Data Frame, then decoded for HTML special characters, and then token blocking is applied. Finally, the output is printed, displaying Blocking Keys and corresponding entities in a neater, better-readable way. Firstly, a subset of 10 mid-sized blocks is printed and then the entire resulting blocking index is stored in the $blocking\_index\_print.txt$ file to provide easy access for later reference. The process captures the output using the $redirect\_stdout$ context manager, ensures data integrity, and confirms task completion.

Task B

Task B involves computing the necessary comparisons to resolve duplicates within the previously created blocks. For this purpose, a dedicated function called $calculate\_comparisons$ is developed. This function accepts the blocking index as input and is designed to determine the total number of pairwise comparisons needed for the entities within the provided blocking data structure.

To calculate the total comparisons, the function applies the combination formula (utilizing the $comb$ function from the $math$ module). It iterates through the sets of IDs (values) in the blocking index, assessing the number of unique entities in each block by calculating the length of these sets. For every block, the function determines the number of possible pairwise comparisons among its entities using the combination formula. This mathematical approach counts the distinct ways to select two entities from the block without repetition, representing the feasible internal comparisons.

By accumulating the pairwise comparisons across all blocks, the function derives the final total of required comparisons for the entire blocking data structure. After executing the $calculate\_comparisons$ function with the given blocking index, the outcome is presented as a printout indicating the total number of pairwise comparisons (resulting in 392,309,099 comparisons). This calculation serves as a vital foundation for comprehending the range of comparisons needed for the task at hand.

Task C

Task C involves the creation and analysis of a Meta-Blocking Graph using the block collection formed in Task A. The Meta-Blocking Graph aims to restructure a redundancy-positive block collection into a new one that contains a substantially lower number of redundant and non-matching comparisons, while maintaining the original number of matching ones (ΔPC≈0, ΔPQ»0). The main idea is that common blocks provide valuable evidence for the similarity of entities. The more blocks two entities share, the more similar and the more likely they are to be matching (Alexiou, 2023, pp. 48-56). To accommodate the large graph size, the graph is stored in an SQLite database rather than being constructed in memory using libraries like $NetworkX$, because when executing the latter, after 30 minutes of run-time, a memory error was being produced as the entire graph could not fit in memory.

The process comprises several phases:

1. Graph Building: A custom class, $SQLiteGraph$, is introduced to manage interactions with the SQLite database for graph representation. This class encompasses various methods for executing different phases of the Meta-Blocking procedure. It features a constructor to establish a connection with the specified SQLite database and the $add\_nodes\_and\_edges\_from\_token\_blocking\_dict(block\_index)$ method, which lasts for 10-20 minutes (depending on computer's performance) and populates the database with nodes and edges.
2. Edge Weighting: The CBS Weighting Scheme (Common Block Scheme) is applied to the graph's edges using the $apply\_cbs\_weighting\_scheme()$ method. This phase, lasting around 15-30 minutes (depending on computer's performance), attributes weights to edges based on the common blocks shared by entities in each comparison.
3. Graph Pruning: An edge-centric pruning algorithm is executed using the $prune\_edges\_with\_weight\_lower\_than(limit)$ method. Edges with a weight below the threshold (2) are removed from the graph.
4. Block Collecting: The pruned graph transforms into a new block collection. Retained edges create blocks, each represented by a minimum-size block. The final number of comparisons in the new block collection after edge pruning is calculated and displayed (53,626,150 comparisons).

In addition to these phases, the $SQLiteGraph$ class offers methods for querying the graph, counting edges and nodes, closing connections, and more. It serves as a versatile tool for managing the process of constructing and querying the graph in an SQLite database, efficiently handling tasks related to edge insertion, weighting, pruning, and block collection.

The assignment concludes by closing the connection to the SQLite database established by the $SQLiteGraph$ class, marking the successful completion of Task C. This comprehensive approach to creating and analyzing a Meta-Blocking Graph supports robust data analysis and resolution of entity duplicates.

Task D

Task D involves the creation of a function, $jaccard\_similarity$, to determine the Jaccard similarity between two entities' titles within a Data Frame. The function takes a Data Frame and two entity IDs (id1 and id2) as inputs. It extracts the titles of the entities linked to the given IDs from the Data Frame. The titles are then transformed to lowercase and tokenized into sets of words. The function, then, computes the intersection of the two sets (common words) and the union of the sets (total unique words). These values are vital for the subsequent calculation of Jaccard similarity. By dividing the count of common words (intersection) by the count of total unique words (union), the Jaccard similarity is computed. This ratio quantifies the similarity between the entities' titles, generating a value between 0 (no similarity) and 1 (full similarity). The calculated Jaccard similarity value is returned as the function's output. This value reflects the degree of similarity between the titles of the two entities, offering a measure of title-based resemblance. Finally, an application of the function is showcased by calculating the Jaccard similarity between entities with IDs 10 and 810 in the provided Data Frame (df).

Code Reproducibility

Ensuring the reproducibility of the results presented in this report is of paramount importance. To facilitate the readers' ability to reproduce the outcomes, the following steps provide guidance on accessing, setting up, and executing the code.

1. Accessing the Code: The complete code used for Tasks A, B, C, and D is available in the child folders '$Code$' and '$Jupyter$' of the root folder '$f2822217$' (my AUEB student ID). Readers are encouraged to download the code files from the provided source.
2. Environment Setup: Depending on the specific tasks and functions, certain libraries and dependencies are required. Ensure that you have the necessary libraries installed.
3. Executing the Code: The code can be executed in a $Jupyter\ Notebook$ for the '$ipynb$' file or any Python environment for the 'py' file. Open the respective code file for each task and follow the instructions within the comments.
4. Task-Specific Instructions: For the assignment's Tasks, refer to the corresponding sections in the $Jupyter\ Notebook$ code or the exported pdf file (if not able to run the '$ipynb$' file) for an in-depth explanation of the code and the methodology used. This report presents only a summary justification of the methodology and code used. The code is designed to be modular and organized, making it straightforward to follow along and reproduce the results.
5. Note: Make sure that the '$ER - Data.csv$' file is placed in the '$Data$' directory before running the code.

By following these steps, readers can confidently reproduce the presented results and gain a deeper understanding of the methodologies applied in this study.

References

Alexiou, G. (2023, July 5). Entity Resolution. Athens, Attica, Greece: AUEB's MSc Business Analytics.