Part 3: Blocking

Clarence Cheung {kcheung6@wisc.edu}
Jin Ruan {jruan3@wisc.edu}

Pre-processing

• Understanding the data

In the development process, we look at some useful attributes such as ***zipCode, streetAddress, name*** and ***city*** to do blocking. We first figure out the amount of missing values in each attribute and the table below shows our investigation results:

	zipCode	*city*	*name*	*streetAddress*
table A	*missing*	*no missing*	*no missing*	*missing*
table B	*no missing*	*no missing*	*no missing*	*missing*

From the above table, ***city*** and ***name*** seem to be good choices for the blocking process as there are no missing values. Although there are some missing values in ***zipCode***, this attribute is still considered a good option since a small amount of 18 tuples will not hinder us from obtaining a reasonable candidate set.

• Cleaning the data

After learning the data and experimenting with the blockers, we did some modification to our original tables. We modified the schema naming so that more feature functions are usable in the rule-based blocker. We converted all the values in ***name*** to lowercase because it caused a bad performance on *Jaccard Measure*. Then, we "normalized" some values in ***city*** so that there is no variation. For example, *Brooklyn* is converted to *New York* since the conversion will prevent more promising matching tuples from being eliminated.

Development

In the development stage, we followed some of the suggestions from the ***Magellan User Manual*** and came up with three different trials.

• First attempt:

Initially, we used the **Attribute Equivalence** blocker in **Magellan** on ***zipCode***. As there are a few missing values in table A, we first splited the table into two smaller table in which one has ***zipCode*** and the other has ***NaN***. Then we applied the blocker and check the results with the debugger. Unfortunately, some tuple pairs are eliminated from the candidate set when they actually refer to the same entity. Since the same restaurant has different ***zipCode***s labeled from different websites, we decided that blocking on ***zipCode*** with Attribute Equivalence is not promising enough to generate a good candidate set.

• Second attempt:

Our second attempt began with using **Attribute Equivalence** blocker on the attribute ***city*** between table A and table B to get the candidate set C. Then we performed an *overlapping* with $size = 1$ on ***name*** in C. To be conservative, the debugger was used again and unexpected results happened. For example, the overlap blocker eliminated $\langle jinwei, jin\ wei \rangle$ because there is no overlapping.

• **Final attempt:**

Learning from previous experience, we first started with the **Attribute Equivalence** blocker on attribute ***city*** to obtain a table C, and applied the rule-based blocker on C. ***Jaccard Measure*** with a threshold value of 0.4 is chosen for the *Similarity Function* in the rule-based method (We did not consider ***TF/IDF*** because it is too slow in our initial trials.) However,the debugger prompted us many false-negatives and so we decreased the threshold value to 0.2, and obtained the candidate set E. A further investigation on set E from the debugger hinted us that almost all of the false-negatives have exact ***streetAddress*** values. This information leaded us to apply an **Attribute Equivalence** blocker on ***streetAddress*** from the eliminated tuple pairs. Since it takes hours to do rule-based blocking on ***name***, a faster alternative is to first equalize the attribute ***streetAddress*** between table A and table B to reduce candidate tuple pairs and get table D. Then we used the rule-based method with ***Jaccard Measure*** on ***name***, but blocked tuple pairs with similarity-value ≥ 0.2 to get table F. The final step is combining table E and F via union to obtain the candidate set G (*which has approx.* $430,000$ *tuple pairs.*) A workflow graph can be found in the *Appendix* section.

Debugger

As mentioned above, we utilized the debugger frequently in our iterative development process. We think the debugger is very useful in the sense that it tells us the performance for each blocker. Without it, it will be hard for us to realize the problem of using **Attribute Equivalence** blocking on ***zipCode***. However, in an iterative manner of development, it is extremely painful to use the current version of debugger because the time to debug is quite long. In our case, the debugging process sometimes takes more than half an hour to complete. From our observation, the current debugger uses the original tables, which are huge in size, as input parameters. We hope that the future version of the debugger can be more flexible in the way that it can support sub-tables as the users only want to compare the new candidate set with candidate sets in certain steps. In that way, the time for debugging can possibly be reduced.

Miscellaneous

• **Time spent:**

We believe the time spent on this stage is roughly 1 day and more since it takes some time to run the rule-based blocker and the debugger.

• **Size of candidate set:**

There are 9947 tuples in table A and 28787 tuples in table B. The numbers give us a Cartesian product of roughly 280 *million* tuple pairs. Applying the blockers that we mentioned above, the tuple pairs are reduced to 20 *million* in the intermediate steps and a final reduction to roughly $400,000$ pairs. We believe this number is promising enough for use in the next stage.

Other issues

The availability of functions and the limited functionality to manipulate tables in **Magellan** has a big impact on our time spent in the project. In our first attempt, we would like to split a single table based on

certain attribute but **Magellan** currently does not support operations within a table. We have to create our own functions to split the table, and then manually reset the splited tables to the valid format. Other than that, the **combine_outputs_via_union** function is not robust enough to handle tables of all kind, and it would be great if this function can be used by the split tables. Also, it would be appreciated if some of the functions can provide "Estimated Time Remaining (ETA)" to hint the users.

Appendix

The figure below shows a rough sketch on our workflow in the final blocking process: