CS 839: DATA SCIENCE

Project Stage 3 - Report

Team Members

● Arun Jose (jose4@wisc.edu)
● Kundan Kumar (kkumar36@wisc.edu)
● Sushma Kudlur Nirvanappa(kudlurnirvan@wisc.edu)

Initial Data from CloudMatcher

Candidate Set: 27424
Prediction List: 140
Table A: 4885
Table B: 5396

Methodology followed

Since our candidate set C had 27424 (> 500) pairs, we proceeded to calculate the density in order to reduce our candidate set and determine precision and recall.

1. We manually labelled a random sample of 50 pairs from C (stored in B0_Labeled.csv) and found it to be 9/50 true values.
2. Since this was less than 20 percentage, we wrote a blocking rule to reduce the candidate set to more accurate matches. We used the *py_entitymaching* package to create a feature with a Jaccard of 3-gram on Movies Names and created a rule to block pairs with similarities less than 0.5 to be blocked.
3. This reduced the candidate set from 27424 to 491 tuples.
4. We ran the *'debug_blocker'* module to ensure that we are not dropping any true matches by using the new blocking rule we have written to reduce the candidate set. Since we didn't see many matches in the output of '*debug_blocker*' from the candidate set, we decided to proceed.
5. Next, we again randomly sampled 50 pairs from C, manually labelled these (stored in B1_Labeled.csv) and found it to be 19/50 in favour of True.
6. Since we got a density of more than 20 percentage, we terminated the iteration.

7. We randomly sampled a further 328 tuples and labelled them as matches/non-matches via a label True/False. As a result, we now had 428 labelled pairs (stored in labeled_pairs.csv) to compute the accuracy of the candidate set C. The labelled file has the following header structure: 'id1,id2,label'.
8. The labelled tuples were then used to calculate precision and recall using the '*estimate_PR*' module in the code.

Blocking Rule

For the Blocking step, we decided to use a Jaccard Measure over a 3-gram tokenized version between the names from the candidate tuple pairs and set a threshold of 0.5 to decide whether to block or not. The code used has been mentioned below:

```
block_f = em.get_features_for_blocking(dfa, dfb, validate_inferred_attr_types=False)
block_f = block_f.iloc[[4],:]
rb = em.RuleBasedBlocker()
rb.add_rule(['Name_Name_jac_qgm_3_qgm_3(ltuple, rtuple) < 0.5'], block_f)
```

Final Precision & Recall

- Precision: [0.983121531366644 - 0.9932796485743588]
- Recall: [0.9828007156935782 - 0.9925687424345006]

Note: While running the function '*run_debug_blocker*', we encountered some interesting observations. Even while running 'run_debug_blocker' for the candidate set from CloudMatcher gave us a lot more matches to be dropped than expected. We came to the conclusion that the CloudMatcher blocking rule must be playing a role in those (actual) matches getting dropped out of predicted matches. To confirm this, we had a discussion with Yash (TA), and he confirmed the same. He stated that there are a few scenarios which CloudMatcher doesn't handle like numerical values which were a reason that 'Year' wasn't getting picked up as a feature for us. He also added that we were required to clean our data in early stage 3 (which we weren't aware and it wasn't mentioned in our project page - learning for the future :)). He suggested an approach to circumvent this issue. As per his suggestion, we ran 'run_debug_blocker' on the initial candidate set and got the top 500 matches that were dropped. We manually filtered 266 tuples out of these 500 which were actually matches and should have been in prediction set and candidate set. We added these 266 tuples in candidate set and prediction set and then followed the methodology discussed above.