CS 839 Stage 3 Report

Entity Matching (EM) about Books

Xiuyuan He, Chenlai Shi, Mingren Shen

1.Description of Entity Type

We want to match data science books in library of UW-Madison and UIUC. The book information of UW-Madison is from here and the book information of UIUC is from here. Details can be found from our Stage 2 Report here.

For each table, The tuples of contains one ID column and seven property columns extracted include:

1) Title: book title, string
2) Author: book authors, string
3) Publication:information of publishers, string
4) Format: book types, category, e.g. journal, magazines
5) ISBN: Search Results International Standard Book Number,integer
6) Series: book series information, string
7) Physical Details:Physical description in book cataloging, string

We name the table about UW-Madison ***TableA.csv*** and the table about UIUC ***TableB.csv***. And there are

- 4824 tuples in table ***TableA.csv***
- 5060 tuples in table ***TableB.csv***

2.Blocker

We first downsample Table A and B to 1000 tuples each. For the same book, since we got the data from two different library websites, their attributes may not be the exact same. Therefore, we applied an Overlap Blocker over some of the attributes, including the Title, Author,

Publication and Series of the book. After multiple tests, we found the best overlap_size for each attribute.

And our final blocker is :
(1) Book author names with 2 words overlapped(1307 tuples left)
(2) Book title with 4 words overlapped(606 tuples left)

After the blocker step, we get set G with 300 tuple pairs.

3. Sample and training

From the result of blocking, we obtained set_C and further randomly sampled 300 tuple pairs and manually labeled them, named G. We split G into I,training set, and J, test set. I contains 200 randomly assigned tuples while J contains the remaining 100.

In the feature engineering process, we first exclude the features that are automatically generated by ID and ISBN, and add Jaccard scores on Book Title, Author, Publication and Series with space as the delimiter. We fit 19 features into Decision Tree, Random Forest, SVM, Naive Bayes, Logistic Regression, Linear Regression with 5-fold cross-validation.

Matcher	Average precision	Average recall	Average f1
Decision Tree	0.878	0.814	0.892
Random Forest	0.923	0.884	0.911
SVM	0.797	0.711	0.761
Naive Bayes	0.779	0.869	0.808
Logistic Regression	0.859	0.817	0.833
Linear Regression	0.886	0.924	0.910

After first round of debug, we found that the learning method had difficulty in distinguishing books/journals in different versions/edition/conference. So we introduced a new feature that

capture this piece of information embedded in book titles. Basically, we extracted the roman numerals from the title and use 1-0 to indicate whether they are the same from each tuple pair.

Matcher	Average precision	Average recall	Average f1
Decision Tree	0.880	0.896	0.897
Random Forest	0.935	0.882	0.851
SVM	0.847	0.679	0.737
Naive Bayes	0.779	0.872	0.786
Logistic Regression	0.844	0.841	0.849
Linear Regression	0.903	0.953	0.897

3. Prediction on J

Matcher	Precision	Recall	F1
Decision Tree	0.926	0.806	0.862
Random Forest	**0.938**	**0.968**	**0.952**
SVM	0.677	0.677	0.677
Naive Bayes	0.675	0.871	0.761
Logistic Regression	0.844	0.871	0.857
Linear Regression	0.903	0.903	0.903

So the best matcher that we found is random forest with precision 0.938, recall 0.968, f1 score 0.952. We actually managed to achieve high recalls, since attributes of books, like title and author, are easy to generate different similarity features and eventually the matching tuple pairs are recognizable based on our features.

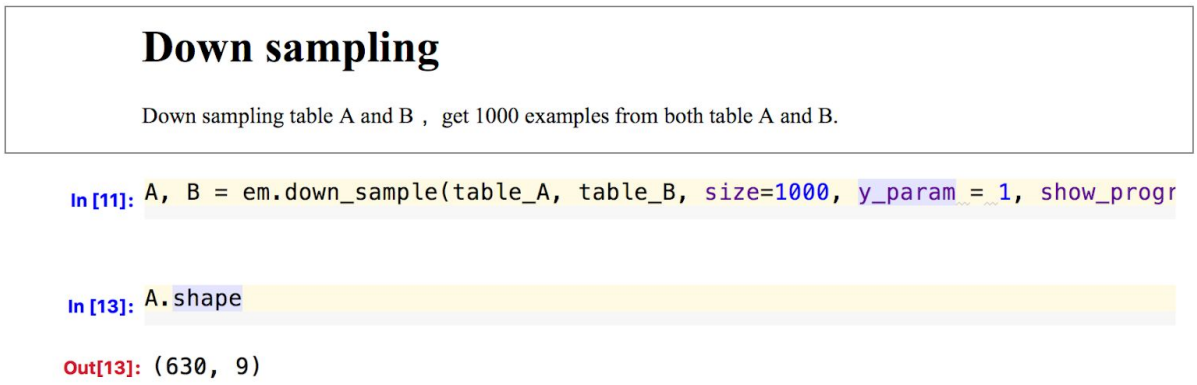
Approximate time estimates:
(a) blocking: 10 mins

(b) labeling data: 60 mins
(c) finding the best matcher: 60 mins

5. Comments on Magellan

We modified the down_sample function so that it won't return unpredicted number of samples from the data set.

For example, when we want to sample 1000 examples from Table_A and Table_B using the down_sample function. However the results are not as desired:



According to the document, which says "`table A should be close to size * y_param.`" however, we can clearly see, that the code returns only 630 results when the expected results should be close to `size (1000 ) * y_param (1) = 1000`.

So we further check the source code, the `down_sample.py` and find that the reason is that the author want to enhance the number of matched entity instead of just random sampling and the key function is `_probe_index(table_b, y_param, s_tbl_sz, s_inv_index, show_progress=True, seed=None)` . And this function using `h_table = set()` to collect the sampled index. However, set does not allow duplicate keys and this makes the final result of sampled results from TableA is less than what we expected.

So here we propose a modified version of the code that can realize the exact number where we keeps inserting the next item from positive candidates list if the current item is already in the result set. And the full version code can be found here.

We understand the purpose of `down_sample.py` is to make the matched tuple pairs more likely to appear in the sampled results. However, for our data set we find the differences is a little bit

too large(nearly 40% items less than expected). So we use our modified `down_sample.py` to do downsample.

```
# Modified by us
# h_table.update(smpl_pos_neg)
for itemNN in smpl_pos_neg:
    while itemNN in h_table:
        itemNN = sorted_key_values[num_pos][0]
        num_pos += 1
    h_table.add(itemNN)
```