Extract Structed Data of Books from Amazon and Barnes & Noble

Lan Bai, Chaoqun Mei, Yuzhe Ma

April 15, 2018

Abstract In this project stage, we used the py_entitymatching to match data of books from Amazon and Barnes & Noble in Jupyter Notebook. We finally chose Decision Tree to be the matcher algorithm and finally achieved 98.04% precision and 98.04% recall on the test set.

1. Entities to match

The source of two tables (A, B) are from Amazon books and Barnes & Noble books. We narrowed down book categories to fiction, health, history, business and science books when crawling data from the web. For each tuple, the attributes are:

ID: Number ID of book in the table, from 1 to N (number of tuples) .

Publisher: The book's publisher.

Time: The book's publish date.

Author: The book's author's name.

Title: The book's title (name).

2. Blocker

We use an AttrEquivalenceBlocker and an OverlapBlocker. We assume that the same books should have same publish date, so we use AttrEquivalenceBlocker to block tuple pairs whose Time attributes are not the same (We let the Time to be the same format in preprocessing). We also assume that the same books should have overlap of at least one word in their title, so we use OverlapBlocker to block tuples whose Title attributes have less than 1-word overlap.

We have 437 tuple pairs after blocking

3. Number of tuple pairs in the sample G

We labeled 350 tuple pairs in G.

4. Precision, recall, and F-1 of matchers after cross validation

Precision, recall, and F-1 of 6 matchers are provided in the following form:

	Matcher	Average precision	Average recall	Average f1
0	DecisionTree	0.974798	0.968133	0.971320
1	Random Forest	0.963245	0.981036	0.971721
2	SVM	0.743575	0.973735	0.841997
3	Naive Bayes	0.966415	0.910392	0.936425
4	Logistic Reg	0.938431	0.980186	0.957903
5	Linear Reg	0.963603	0.980186	0.971268

5. The best matcher selected

We selected Decision Tree, whose precision is the highest and F1 is the second highest, to be the best matcher. For cross validation the precision is 0.974798, recall is 0.968133 and F1 is 0.971320.

As we have already got high precision and recall, no debugging is needed.

6. Accuracy of the matcher on test set

Precision : 98.04% (100/102)

Recall : 98.04% (100/102)

F1 : 98.04%

False positives : 2 (out of 102 positive predictions)

False negatives : 2 (out of 38 negative predictions)

7. Time of blocking, label the data, find the best matcher

We spend 2-3 hours on blocking. Time of running blocker to blocking tuple is 0.774514s.

We spend 3 hours on labeling.

We spend 2-3 hours on finding the best matcher. Time of running cross validation is 62.990848s.

8. Discussing of getting a higher recall

We have achieved high enough recall (98.04%) in our learning task, where 2 out of 102 positive examples are labeled negatively. In general, one reason why recall could be low is the unbalanced labels in the training set, e.g., when most of the data are negative, then our machine learning algorithms will tend to predict more negative examples, resulting in low recall. Thus to

increase the recall, one can do better sampling such that the labels are more balanced. Another reason is the inappropriate learning algorithm that's being used, thus one needs to try more algorithms and then pick the best one. A third reason is because of the nature of the data itself, e.g., when the ground truth is that most data are negative, then in order to achieve high accuracy, positive data are sacrificed (predicted negatively), which will lead to extremely low recall on the test data. In this case, one can use different measurement to evaluate the model when solving for the learning process, e.g., instead of maximizing accuracy on training data, one can use recall directly as the objective and tries to maximize the recall of the learned model. However, this could possibly decrease other evaluation scores such as the precision.

Bonus points: Comments on Magellan

Magellan is easy to use, especially in blocking and feature construction. We like the feature construction part that construct features automatically.

However, we also found several problems:

1. The install process on Windows:

   I am using Windows 10, and python 3.6 and the install process is not very friendly to user.

   Here is the problems I encountered when install Magellan on Windows 10 and how I solve them:

   I first tried to install Magellan by conda. After I ran the install command, and used **conda list** to check the package is installed, I imported py_entitymatching in python and got ModuleNotFoundError:

```
(py36) C:\Users\bailan>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import py_stringmatching
>>> import py_entitymatching
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'py_entitymatching'
>>> exit()
```

   Interestingly, I can import another package also from uw_magellan called py_stringmatching.

   I tried to install with pip, and then got the same error. I also tried install with source code, the error I got when import py_entitymatching was:

```
(py36) C:\Users\bailan\py_entitymatching>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import py_entitymatching
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\bailan\py_entitymatching\py_entitymatching\__init__.py", line 42, in <module>
    from py_entitymatching.debugblocker.debugblocker import debug_blocker
  File "C:\Users\bailan\py_entitymatching\py_entitymatching\debugblocker\debugblocker.py", line 19, in <module>
    from py_entitymatching.debugblocker.debugblocker_cython import \
ModuleNotFoundError: No module named 'py_entitymatching.debugblocker.debugblocker_cython'
>>> exit()
```

I tried to install an earlier version of py_entitymathing (0.2.0). Then I could import, however there is error when reading the csv file.

I finally solve this problem by upgrading py_entitymatching 0.2.0 to 0.3.0 in **Administer mode** in Anaconda prompt. So I think when installing with conda on windows, run command line or Anaconda prompt in Admininster mode will be better, and I think this should be mentioned in the document because usually we do not need to do so.

2. The confusion in document of py_entitymatching

   In py_entitymatching doc, sometimes a function or a variable is used before it is explained in the following section.

   For example, *block_f* is first used in Rule-based Blocker section, but is not explained until Getting a set of features, which is several sections later.