CS 784 DATA MODELS PROJECT

Department of Computer Science
University of Wisconsin – Madison

Shuang Wu
Table of Contents

Workflow ...................................................................................................................................... 2

Plan ............................................................................................................................................. 2
  Debugging Iteration 1 .............................................................................................................. 2
    Important Note for Debugging Iteration 1 ............................................................................ 3
  Debugging Iteration 2 .............................................................................................................. 3
    Important Note for Debugging Iteration 2 ............................................................................ 4
The Basic Strategy for Entity Matching ...................................................................................... 4
  Special Rule for Title ............................................................................................................... 5
Some False Positive and False Negative Case ...................................................................... 6

Result .......................................................................................................................................... 8
  Six Learning Methods 1st time Cross Validation ................................................................. 8
  1st Learning Based Matcher Selected ................................................................................... 9
Debugging Iteration 1............................................................................................................. 9
  1st Round by random_state = 0 on Splitting G ................................................................. 9
  2nt Round by random_state = 50 on Splitting G ............................................................. 9
  3rd Round by random_state = 120 on Splitting G ............................................................ 11
Overall Improvement After Debugging Iteration 1 ............................................................. 12
Debugging Iteration 2............................................................................................................. 12
  Add Feature Vector ............................................................................................................. 13
  Add exact_match For title and author .................................................................................. 14
  Add year_match .................................................................................................................... 14
  Add one_author_match ........................................................................................................ 14
  Add Special Rule For title .................................................................................................. 15
  Create Positive and Negative Rules .................................................................................... 16
  Test and Compare Each Rule’s Effect on Random Forest .................................................... 16
  Test and Compare Each Rule’s Effect on Naive Bayes ......................................................... 17
  Important Note for Adding Rules ....................................................................................... 18

Comparison .................................................................................................................................. 18
  Random Forest ...................................................................................................................... 19
  Decision Tree ........................................................................................................................ 19
  Support Vector Machine ....................................................................................................... 20
  Naive Bayes .......................................................................................................................... 20
  Linear Regression .................................................................................................................. 21
  Logistic Regression .............................................................................................................. 21
  Final Best Matcher Result ..................................................................................................... 22

Misc ........................................................................................................................................... 22
  Approximate Time Estimation ............................................................................................... 22
1. Workflow

![Workflow diagram]

2. Plan

**Debugging Iteration 1**: Repeating (1 → 2 → 3) until no more FN and FP cases corresponding to ambiguity and incorrect label data appears

# Split table G using random_state = 0, DO first cross validation on table H

# Remove error tuple pairs, split table G and table I using random_state = 0
  # Split H using random_state = 0, do step2 and repair mislabeled data
  # Split H using random_state = 1, do step2 and repair mislabeled data
  # Split H using random_state = 2, do step2 and repair mislabeled data
  # Split H using random_state = 3, do step2 and repair mislabeled data
  .
  .
  .

5. Finally evaluate J using Y*

1. Select best ML matcher Y form using 5 fold CV on H

3. Relabeling, removing incorrect tuple pairs until all FP and FN corresponding to mislabeled tuple pairs are eliminated

2. Train on U and debug on V

4. Add rules on Y until eliminating all False Positive and False Negative appeared during debugging
# Remove error tuple pairs, split table G and table I using random_state = 20
  # Split H using random_state = 0, do step2 and repair mislabeled data
  # Split H using random_state = 1, do step2 and repair mislabeled data
  # Split H using random_state = 2, do step2 and repair mislabeled data
  # Split H using random_state = 3, do step2 and repair mislabeled data
  .
  .
  .

# Remove error tuple pairs, split table G and table I using random_state = 30
  # Split H using random_state = 0, do step2 and repair mislabeled data
  # Split H using random_state = 1, do step2 and repair mislabeled data
  # Split H using random_state = 2, do step2 and repair mislabeled data
  # Split H using random_state = 3, do step2 and repair mislabeled data
  .
  .
  .

# Split finalized table G using random_state = 0 again, do the second cross validation
  on table H to show the overall accuracy improvement by cleaning golden table G.

important note for Debugging Iteration 1:
  a. In the process of resolving data ambiguity and incorrect label issues, I do the iteration
     broadly on table G instead of just splitting locally on table H. This will to a large extent
     ensure correct data on both table J and table I in order to reduce number of FP and FN
     tuple pairs on final table J evaluation.
  b. Using different random_state during splitting table G and table H will enable us to see
     all the FP and FN cases corresponding to data ambiguity and incorrect label.

Debugging Iteration 2: Repeating (1 → 2 → 4) until no more FN and FP cases appears

  # Split H using random_state = 0, do step2 and add rule 1 to matcher Y
  and do CV on the table H to compute the matcher’s accuracy
  # Split H using random_state = 100, do step2 and add rule 2 to matcher Y
  and do CV on the table H to compute the matcher’s accuracy
  # Split H using random_state = 200, do step2 and add rule 3 to matcher Y
  and do CV on the table H to compute the matcher’s accuracy
  .
  .
  .
important note for Debugging Iteration 2:

a. The most confusing part in this matching scenario is, many books (with the same title) might have many different versions. For those cases, I treat different versions of the book as different books. As for some very ambiguous pairs, I went a third party (Amazon and noble & barnes) to check if they really match

b. But first of all, all the book pairs with different “title” are treated as different books. However, the challenge is how can we say the the “title” are different. Three common cases are given below:

   case 2 - Golfâs Finest Par Threes VS. Golfs Finest Par Threes
   case 3 - Suarez â 2016 Updated Edition VS. Ronaldo â 2016 Updated Edition

Virtually the case 1 is not matching pair but the case 2 is a matching pair. Thus we can not just simply use rules such as
<not match if 'title_title_jac_ggm_3_ggm_3(ltuple, rtuple) < 0.9' is true>
to improve accuracy.

c. Based on observations from a and b above, I designed a more logical rule to make decision. The basic strategy is:

```
1. title
   exact_match
   special_rule

2. author
   exact_match
   one_author_match

3. length
   exact_match

4. date
   year_match

Positive rule:
any three of four match --- > match(1)

Negative Rule:
any two of four don't match --- > non-match(0)
```
Special rule for title:

More specifically for a special case (title):
Finally this trigger will solve some issues like those:
<table>
<thead>
<tr>
<th>Left Tuple</th>
<th>Value</th>
<th>Right Tuple</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id</td>
<td>2985</td>
<td>record_id</td>
<td>10067</td>
</tr>
<tr>
<td>date</td>
<td>Apr 16, 2010</td>
<td>description</td>
<td>Do you have what it takes?</td>
</tr>
<tr>
<td>language</td>
<td>English</td>
<td>price</td>
<td>18.0</td>
</tr>
<tr>
<td>rating_value</td>
<td>nan</td>
<td>review_count</td>
<td>nan</td>
</tr>
<tr>
<td>price</td>
<td>14.99</td>
<td>title</td>
<td>Wilderness Survival</td>
</tr>
<tr>
<td>author</td>
<td>Michael Pawluker</td>
<td>rating_value</td>
<td>nan</td>
</tr>
<tr>
<td>rating_star</td>
<td>0.0</td>
<td>author</td>
<td>Mark Ellbrock, Michael Pawluker</td>
</tr>
<tr>
<td>seller</td>
<td>The McGraw-Hill Companies, Inc.</td>
<td>length</td>
<td>288</td>
</tr>
<tr>
<td>short_description</td>
<td>nan</td>
<td>short_description</td>
<td>Living Off the Land with the Clothes on Your Back and the Knife on Your Belt</td>
</tr>
<tr>
<td>length</td>
<td>288</td>
<td>genre</td>
<td>Outdoors</td>
</tr>
<tr>
<td>genre</td>
<td>Outdoors</td>
<td>page_id</td>
<td>4584657162</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Left Tuple</th>
<th>Value</th>
<th>Right Tuple</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Jul 01, 2014</td>
<td>description</td>
<td>To those who went to the War straight from school and survived it, the problem of what...</td>
</tr>
<tr>
<td>language</td>
<td>English</td>
<td>price</td>
<td>17.5</td>
</tr>
<tr>
<td>title</td>
<td>Snow on the Equator</td>
<td>date</td>
<td>September 2015</td>
</tr>
<tr>
<td>rating_value</td>
<td>nan</td>
<td>review_count</td>
<td>nan</td>
</tr>
<tr>
<td>price</td>
<td>5.99</td>
<td>title</td>
<td>Snow on the Equator</td>
</tr>
<tr>
<td>author</td>
<td>H.W. Tillman &amp; Jim Perrin</td>
<td>rating_value</td>
<td>nan</td>
</tr>
<tr>
<td>rating_star</td>
<td>0.0</td>
<td>author</td>
<td>H.W. Tillman, Sir Chris Bonington</td>
</tr>
<tr>
<td>seller</td>
<td>The Perseus Books Group, LLC</td>
<td>length</td>
<td>300</td>
</tr>
<tr>
<td>short_description</td>
<td>nan</td>
<td>short_description</td>
<td>Mount Kenya, Kilimanjaro and the great African odyssey</td>
</tr>
<tr>
<td>length</td>
<td>300</td>
<td>genre</td>
<td>Mountaineering</td>
</tr>
<tr>
<td>genre</td>
<td>Mountaineering</td>
<td>page_id</td>
<td>886714427</td>
</tr>
</tbody>
</table>
3. Results

a. For each of the six learning methods for the first time for these methods on I(H):
split table G with random_state = 20

<table>
<thead>
<tr>
<th>Name</th>
<th>Matcher</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DecisionTree &lt;magellan.matcher.dm matcher, DTMatcher object a...</td>
<td>5</td>
<td>0.916667</td>
<td>0.875000</td>
<td>0.937500</td>
<td>0.869524</td>
<td>1.000000</td>
<td>0.907738</td>
</tr>
<tr>
<td>1</td>
<td>RF &lt;magellan.matcher.rf match e, RFMatcher object a...</td>
<td>5</td>
<td>0.916667</td>
<td>0.933333</td>
<td>0.933333</td>
<td>0.947368</td>
<td>1.000000</td>
<td>0.946140</td>
</tr>
<tr>
<td>2</td>
<td>SVM &lt;magellan.matcher.svm matcher, SVMMatcher object a...</td>
<td>5</td>
<td>0.640000</td>
<td>0.623535</td>
<td>0.623529</td>
<td>0.900000</td>
<td>1.000000</td>
<td>0.889176</td>
</tr>
<tr>
<td>3</td>
<td>NB &lt;magellan.matcher.nb matcher, NBMatcher object a...</td>
<td>5</td>
<td>0.880000</td>
<td>0.937500</td>
<td>0.862353</td>
<td>0.894736</td>
<td>1.000000</td>
<td>0.918918</td>
</tr>
<tr>
<td>4</td>
<td>LogReg &lt;magellan.matcher.logreg matcher, LogRegMatcher object a...</td>
<td>5</td>
<td>0.913043</td>
<td>0.937500</td>
<td>0.823529</td>
<td>0.900000</td>
<td>1.000000</td>
<td>0.914818</td>
</tr>
<tr>
<td>5</td>
<td>LinReg &lt;magellan.matcher.linreg matcher, LinRegMatcher object a...</td>
<td>5</td>
<td>0.840000</td>
<td>0.823535</td>
<td>0.937500</td>
<td>0.947368</td>
<td>0.933333</td>
<td>0.908111</td>
</tr>
</tbody>
</table>

**recall:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Matcher</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DecisionTree &lt;magellan.matcher.dm matcher, DTMatcher object a...</td>
<td>5</td>
<td>1.000000</td>
<td>0.823529</td>
<td>0.937500</td>
<td>0.944444</td>
<td>0.926571</td>
<td>0.926809</td>
</tr>
<tr>
<td>1</td>
<td>RF &lt;magellan.matcher.rf matcher, RFMatcher object a...</td>
<td>5</td>
<td>1.000000</td>
<td>0.823529</td>
<td>0.937500</td>
<td>1.000000</td>
<td>0.926571</td>
<td>0.926809</td>
</tr>
<tr>
<td>2</td>
<td>SVM &lt;magellan.matcher.svm matcher, SVMMatcher object a...</td>
<td>5</td>
<td>0.954545</td>
<td>0.882353</td>
<td>0.875000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>0.942380</td>
</tr>
<tr>
<td>3</td>
<td>NB &lt;magellan.matcher.nb matcher, NBMatcher object a...</td>
<td>5</td>
<td>1.000000</td>
<td>0.823529</td>
<td>0.937500</td>
<td>0.944444</td>
<td>1.000000</td>
<td>0.952959</td>
</tr>
<tr>
<td>4</td>
<td>LogReg &lt;magellan.matcher.logreg matcher, LogRegMatcher object a...</td>
<td>5</td>
<td>0.954545</td>
<td>0.882353</td>
<td>0.937500</td>
<td>1.000000</td>
<td>1.000000</td>
<td>0.942380</td>
</tr>
<tr>
<td>5</td>
<td>LinReg &lt;magellan.matcher.linreg matcher, LinRegMatcher object a...</td>
<td>5</td>
<td>0.954545</td>
<td>0.882353</td>
<td>0.937500</td>
<td>1.000000</td>
<td>1.000000</td>
<td>0.954880</td>
</tr>
</tbody>
</table>

**f1:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Matcher</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DecisionTree &lt;magellan.matcher.dm matcher, DTMatcher object a...</td>
<td>5</td>
<td>0.956522</td>
<td>0.848485</td>
<td>0.937500</td>
<td>0.871795</td>
<td>0.962963</td>
<td>0.915453</td>
</tr>
<tr>
<td>1</td>
<td>RF &lt;magellan.matcher.rf matcher, RFMatcher object a...</td>
<td>5</td>
<td>0.956522</td>
<td>0.875000</td>
<td>0.903222</td>
<td>0.972973</td>
<td>0.962963</td>
<td>0.934137</td>
</tr>
<tr>
<td>2</td>
<td>SVM &lt;magellan.matcher.svm matcher, SVMMatcher object a...</td>
<td>5</td>
<td>0.893617</td>
<td>0.882353</td>
<td>0.848485</td>
<td>0.947368</td>
<td>1.000000</td>
<td>0.914365</td>
</tr>
<tr>
<td>3</td>
<td>NB &lt;magellan.matcher.nb matcher, NBMatcher object a...</td>
<td>5</td>
<td>0.938170</td>
<td>0.900091</td>
<td>0.900091</td>
<td>0.918919</td>
<td>1.000000</td>
<td>0.934654</td>
</tr>
<tr>
<td>4</td>
<td>LogReg &lt;magellan.matcher.logreg matcher, LogRegMatcher object a...</td>
<td>5</td>
<td>0.933333</td>
<td>0.900091</td>
<td>0.848485</td>
<td>0.947368</td>
<td>1.000000</td>
<td>0.927656</td>
</tr>
<tr>
<td>5</td>
<td>LinReg &lt;magellan.matcher.linreg matcher, LinRegMatcher object a...</td>
<td>5</td>
<td>0.893617</td>
<td>0.882353</td>
<td>0.937500</td>
<td>0.972973</td>
<td>0.965517</td>
<td>0.930392</td>
</tr>
</tbody>
</table>

b. After the first time CV, RF is chosen as best classifier because it has highest accuracy of 94.61%
c. **Debugging Iteration 1 (data ambiguity and incorrect label):**

1st round by random_state = 0 on splitting G

I splits H into U and V five times with different random_state number, and repair the mislabeled tuple pairs on table G. One of the FP and FN cases is like this:

<table>
<thead>
<tr>
<th>Left Tuple</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id</td>
<td>2973</td>
</tr>
<tr>
<td>publisher</td>
<td>Falcon Guides</td>
</tr>
<tr>
<td>date</td>
<td>Feb 01, 2011</td>
</tr>
<tr>
<td>description</td>
<td>Backpacker’s Using a GPS: Digital Trip Planning, Recording, and Sharing is a complete guide to …</td>
</tr>
<tr>
<td>language</td>
<td>English</td>
</tr>
<tr>
<td>title</td>
<td>Backpacker Magazine’s Using a GPS</td>
</tr>
<tr>
<td>rating_value</td>
<td>nan</td>
</tr>
<tr>
<td>price</td>
<td>11.99</td>
</tr>
<tr>
<td>author</td>
<td>Bruce Grubbs</td>
</tr>
<tr>
<td>rating_star</td>
<td>0.0</td>
</tr>
<tr>
<td>seller</td>
<td>The Rowman &amp; Littlefield Publishing Group</td>
</tr>
<tr>
<td>short_description</td>
<td>Digital Trip Planning, Recording, And Sharing</td>
</tr>
<tr>
<td>length</td>
<td>96</td>
</tr>
<tr>
<td>genre</td>
<td>Outdoors</td>
</tr>
<tr>
<td>page_id</td>
<td>938494636</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Right Tuple</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id</td>
<td>5851</td>
</tr>
<tr>
<td>ISBN</td>
<td>9.78076276776e+12</td>
</tr>
<tr>
<td>description</td>
<td>Backpacker magazine’s Outdoor Knots brings you essential mind gear from the two most res…</td>
</tr>
<tr>
<td>price</td>
<td>11.99</td>
</tr>
<tr>
<td>date</td>
<td>February 2011</td>
</tr>
<tr>
<td>publisher</td>
<td>Falcon Guides</td>
</tr>
<tr>
<td>review_count</td>
<td>nan</td>
</tr>
<tr>
<td>title</td>
<td>Backpacker Magazine’s Outdoor Knots</td>
</tr>
<tr>
<td>rating_value</td>
<td>nan</td>
</tr>
<tr>
<td>author</td>
<td>Clyde Soles</td>
</tr>
<tr>
<td>length</td>
<td>96</td>
</tr>
<tr>
<td>short_description</td>
<td>The Knots You Need To Know</td>
</tr>
</tbody>
</table>

both split on G and H with random_state = 0, after 1st round of cleaning, we get

# precision: 97.44% (38/39) → 97.56% (40/41)
# recall: 90.48% (38/42) → 93.02% (40/43)
# F1: 93.83% → 95.24%
# False positive: 1 (out of 39 positive predictions) → 1 (out of 41 positive predictions)
# False negative: 4 (out of 101 negative predictions) → 3 (out of 99 negative predictions)

2nd round by random_state = 50 on splitting G

I splits H into U and V five times with different random_state number, and repair the mislabeled tuple pairs on table G. One of the FP and FN cases is like this:
split on G with random_state = 50 and split on H with random_state = 0, after 2nd round of cleaning, we get

<table>
<thead>
<tr>
<th>Left Tuple</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id</td>
<td>177</td>
</tr>
<tr>
<td>publisher</td>
<td>Triumph Books</td>
</tr>
<tr>
<td>date</td>
<td>Mar 01, 2010</td>
</tr>
<tr>
<td>description</td>
<td>162-0: Imagine a Twins Perfect Season imagines that season by identifying the mo...</td>
</tr>
<tr>
<td>language</td>
<td>English</td>
</tr>
<tr>
<td>title</td>
<td>162-0: Imagine a Twins Perfect Season</td>
</tr>
<tr>
<td>rating_value</td>
<td>nan</td>
</tr>
<tr>
<td>price</td>
<td>11.99</td>
</tr>
<tr>
<td>author</td>
<td>Dave Wright</td>
</tr>
<tr>
<td>rating_star</td>
<td>0.0</td>
</tr>
<tr>
<td>seller</td>
<td>Chicago Review Press, Inc. DBA Independent Publishers Group</td>
</tr>
<tr>
<td>short_description</td>
<td>The Greatest Wins!</td>
</tr>
<tr>
<td>length</td>
<td>304</td>
</tr>
<tr>
<td>genre</td>
<td>Baseball</td>
</tr>
<tr>
<td>page_id</td>
<td>708499380</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Right Tuple</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id</td>
<td>6452</td>
</tr>
<tr>
<td>ISBN</td>
<td>9.78161749074e+12</td>
</tr>
<tr>
<td>description</td>
<td>162-0: Imagine a Red Sox Perfect Season imagines that season by identifying the m...</td>
</tr>
<tr>
<td>price</td>
<td>11.99</td>
</tr>
<tr>
<td>date</td>
<td>March 2010</td>
</tr>
<tr>
<td>publisher</td>
<td>Triumph Books</td>
</tr>
<tr>
<td>review_count</td>
<td>nan</td>
</tr>
<tr>
<td>title</td>
<td>162-0: Imagine a Red Sox Perfect Season</td>
</tr>
<tr>
<td>rating_value</td>
<td>nan</td>
</tr>
<tr>
<td>author</td>
<td>Mark Colman, Tony Massaroti</td>
</tr>
<tr>
<td>length</td>
<td>304</td>
</tr>
<tr>
<td>short_description</td>
<td>The Greatest Wins!</td>
</tr>
</tbody>
</table>

# precision: 97.56% (40/41) → 100.0% (40/40)
# recall: 97.56% (40/41) → 97.56% (40/41)
# F1: 97.56% (40/41) → 98.77%
# False positive: 1 (out of 41 positive predictions) → 0 (out of 40 positive predictions)
# False negative: 1 (out of 99 negative predictions) → 1 (out of 100 negative predictions)
3rd round by random_state = 120 on splitting G

I splits H into U and V five times with different random_state number, and repair the mislabeled tuple pairs on table G. One of the FP and FN cases is like this:

<table>
<thead>
<tr>
<th>Left Tuple</th>
<th>Right Tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id 6480</td>
<td>ISBN 9.78180734509e+12</td>
</tr>
<tr>
<td>publisher Charlesbridge</td>
<td>description Touchdown! These tales from the gridiron will set fans abuzz. Fun, fille...</td>
</tr>
<tr>
<td>date Sep 01, 2009</td>
<td>price 8.99</td>
</tr>
<tr>
<td>description This book hits a grand slam right out of the park! No diehard devotee of the ...</td>
<td>date July 2011</td>
</tr>
<tr>
<td>language English</td>
<td>publisher Charlesbridge</td>
</tr>
<tr>
<td>title Book of Baseball Stuff</td>
<td>review_count nan</td>
</tr>
<tr>
<td>rating_value nan</td>
<td>rating_value nan</td>
</tr>
<tr>
<td>price 8.99</td>
<td>author Ron Martiriano</td>
</tr>
<tr>
<td>author Ron Martiriano</td>
<td>rating_star 0.0</td>
</tr>
<tr>
<td>rating_star 0.0</td>
<td>seller Random House, LLC</td>
</tr>
<tr>
<td>seller Random House, LLC</td>
<td>short_description nan</td>
</tr>
<tr>
<td>short_description nan</td>
<td>length 192</td>
</tr>
<tr>
<td>length 192</td>
<td>genre Baseball</td>
</tr>
<tr>
<td>genre Baseball</td>
<td>page_id 801564884</td>
</tr>
</tbody>
</table>

Split on G with random_state = 120 and split on H with random_state = 0, after 3rd round of cleaning, we get

# precision: 100.0% (38/38) → 100.0% (38/38)
# recall: 97.44% (38/38) → 100.0% (38/38)
# F1: 98.7% → 100.0% 
# False positive: 0 (out of 38 positive predictions) → 0 (out of 38 positive predictions)
# False negative: 1 (out of 102 negative predictions) → 0 (out of 102 negative predictions)

After finishing Debugging Iteration 1, we can see how much overall accuracy improvement by doing CV on H (split on G with random_state = 20 as did the first time CV).
Now RF and NB are the best learning-based matchers.

Debugging Iteration 2 (add rules as triggers on matcher Y):

From the previous part, we already know the so-far best precision/recall/f1 based on the H:
I add short_description and delete price, and then extract feature vectors.

```python
# add one more feature vector from this

feat_table = mg.get_features_for_matching(A, B)

feat_subset_iter1 = feat_table[3:29]
feat_subset_iter2 = feat_table[32:43]
feat_subset_iter3 = feat_table[4:54]

feat_subset_all = feat_subset_iter1.append(feat_subset_iter2)
feat_subset_all = feat_subset_all.append(feat_subset_iter3)

# to this

feat_table = mg.get_features_for_matching(A, B)

feat_subset_iter1 = feat_table[3:29]
feat_subset_iter2 = feat_table[32:50]

feat_subset_all = feat_subset_iter1.append(feat_subset_iter2)
```

Then we could improve recall a little bit from 0.979144 → 0.988235

```python
result = mg.cv_matcher_and_trigger(rf, [], table = H,
                                 exclude_atts=[‘id’, ‘ltab_id’, ‘rtab_id’, ‘gold’],
                                 target_att=’gold’, random_state = 1200)

result[‘cv_stats’]
```

After this, I do CV by all the machine learning algorithm on H, and RF is still the best matcher so far, so I continue to debug on RF...

1st round by random_state = 0 on splitting H
2nd round by random_state = 120 on splitting H
3rd round by random_state = 500 on splitting H

Some cases of FP and FN has been shown in Plan section above. Thus I will directly show you the final results with adding rules during debugging.

add exact_match for title and author:

```python
def get_feature_fn("exact_match(tuple['title'], tuple['title'])", mg_match_t, mg_match_s)
mg.add_feature(feat_table, 'title_title_exm', feature1)

feature2 = mg.get_feature_fn("exact_match(tuple['author'], tuple['author'])", mg_match_t, mg_match_s)
mg.add_feature(feat_table, 'author_author_exm', feature2)
```

add year_match:

```python
# x, y will be of type pandas series e.g. 19-march-15 & march-15
def year_exact_date(x, y):
    if type(x['date']) == int or type(y['date']) == int or type(x['date']) == float or type(y['date']) == float:
        return False

    x_dateSet = x['date'].split(','
    y_dateSet = y['date'].split(

    if len(x_dateSet) > 1 and len(y_dateSet)>1:
        if x_dateSet[1] == y_dateSet[1]:
            return True

    return False

Date_rule = 'match_exact_date'
mg.add_blackbox_feature(feat_table, Date_rule, match_exact_date)
```

add one_author_match:

```python
# x, y will be of type pandas series e.g. 19-march-15 & march-15
import re

def one_author_author(x, y):
    x_dateSet = re.split(r'\[\S\]*', x['author'])
    y_dateSet = re.split(r'\[\S\]*', y['author'])

    if len(x_dateSet) > len(y_dateSet):
        small = x_dateSet
        large = y_dateSet
    else:
        small = y_dateSet
        large = x_dateSet

    smallsize = len(small)
    largesize = len(large)

    for index1 in range(0, smallsize):
        for index2 in range(0, largesize):
            if small[index1].strip() == large[index2].strip():
                return True

    return False

Author_rule = 'match_one_author'
mg.add_blackbox_feature(feat_table, Author_rule, match_one_author)
```
add special rule for title:

```python
def match_title_with_tolerance(x, y):
    # x, y will be of type pandas series

    # get title attribute
    x_title = x['title']
    y_title = y['title']

    x_titleSet = x_title.split(' ')
    y_titleSet = y_title.split(' ')

    # decide which one is shorter, so it can avoid cases like :
    # Wilderness Survival & Wilderness Survival Handbook : Primitive Skills for Short-Term Survival and Long-Term Comfort
    # they are actually the same book, but one of this title is abbreviated
    if len(x_titleSet) > len(y_titleSet):
        longer_string = x_titleSet
        shorter_string = y_titleSet
    else:
        shorter_string = x_titleSet
        longer_string = y_titleSet

    # compare each character in both string
    for index in range(0, len(shorter_string)):
        if shorter_string[index].strip() != longer_string[index].strip():
            word_in_shorter_string = shorter_string[index].strip()
            word_in_longer_string = longer_string[index].strip()

            # volume 101 vs volume 162
            if word_in_shorter_string.isdigit() or word_in_longer_string.isdigit():
                return False

            # if the size of each strings is the same or 1 character longer than the shorter one, do the following
            # bigger than 1 is a trade-off for misspelling e.g. Golf's & Golf
            # in this case, we can only accept one letter incorrect
            else:
                if abs(len(word_in_shorter_string) - len(word_in_longer_string)) > 1:
                    # give up the negative rule directly and return true
                    return True

                if len(word_in_shorter_string) > len(word_in_longer_string):
                    longer_word = word_in_longer_string
                    shorter_word = word_in_shorter_string
                else:
                    shorter_word = word_in_shorter_string
                    longer_word = word_in_longer_string

                count = 0  # count for # of letter not matching
                total = 0

                pointer1 = 0  # pointer for each letter on the shorter word
                pointer2 = 0  # pointer for each letter on the longer word

                # e.g. Géof & Golf
                for index2 in range(0, len(shorter_word)):
                    total += 1
                    if len(word_in_shorter_string) == len(word_in_longer_string):
                        if shorter_word[index2] != longer_word[index2]:
                            count += 1
                    else:
                        while shorter_word[pointer1] != longer_word[pointer2]:
                            count += 1
                            pointer1 += 1
                            if abs(pointer1 - pointer2) > 1:
                                return False

                        pointer1 += 1
                        pointer2 += 1

                # trade-off for error detection, if 2 out of 10 letters
                # in one word differ from each other, then we say they
                # are non-matching
                if count / total > 0.2:
                    return False

    return True
```
After creating positive and negative Rule 1 to Rule 4:

```
# Add trigger - target false positives: use title related feature
pos_trigger1 = mg.MatchTrigger()
pos_trigger1.add_cond_rule(\'match_exact_date(ltuple, rtuple) and title_title_jac_qgm_3_qgm_3(ltuple, rtuple) > 0.8 and author_author_jac(ltuple, rtuple), feat_table\'
pos_trigger1.add_cond_status(True)
pos_trigger1.add_action(1)

pos_trigger2 = mg.MatchTrigger()
pos_trigger2.add_cond_rule(\'match_exact_date(ltuple, rtuple) and match_one_author(ltuple, rtuple) and length_length_exm(ltuple, rtuple) and title_title_jac_qgm_3_qgm_3(ltuple, rtuple) > 0.8, feat_table\'
pos_trigger2.add_cond_status(True)
pos_trigger2.add_action(1)

pos_trigger3 = mg.MatchTrigger()
pos_trigger3.add_cond_rule(\'match_exact_date(ltuple, rtuple) and match_one_author(ltuple, rtuple) and title_title_exm(ltuple, rtuple), feat_table\'
pos_trigger3.add_cond_status(True)
pos_trigger3.add_action(1)

pos_trigger4 = mg.MatchTrigger()
pos_trigger4.add_cond_rule(\'match_exact_date(ltuple, rtuple) and match_one_author(ltuple, rtuple) and title_title_jac_qgm_3_qgm_3(ltuple, rtuple) > 0.5, feat_table\'
pos_trigger4.add_cond_status(True)
pos_trigger4.add_action(1)
```

(1)

Check this out, f1 of CV on H(I) by RF is 98.35% without any rules:

```
result = mg.cv_matcher_and_trigger(rf, [],
    table = B, exclude_attr=\'\_id, \'ltuple.record_id\', \'rtuple.record_id\', \'gold\', target_attr='gold', random_state = 1200)
```

```
<table>
<thead>
<tr>
<th>Metric</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>precision</td>
<td>5</td>
<td>1.00000</td>
<td>0.94444</td>
<td>1.00000</td>
<td>1</td>
<td>0.988889</td>
</tr>
<tr>
<td>1</td>
<td>recall</td>
<td>5</td>
<td>0.95456</td>
<td>1.00000</td>
<td>0.94176</td>
<td>1</td>
<td>0.97144</td>
</tr>
<tr>
<td>2</td>
<td>F1</td>
<td>5</td>
<td>0.97674</td>
<td>0.97142</td>
<td>0.96989</td>
<td>1</td>
<td>0.980574</td>
</tr>
</tbody>
</table>
```

f1 of CV on H(I) by RF is 96.72% with only positive rules:

```
result = mg.cv_matcher_and_trigger(rf, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4],
    table = B, exclude_attr=\'\_id, \'ltuple.record_id\', \'rtuple.record_id\', \'gold\', target_attr='gold', random_state = 1200)
```

```
<table>
<thead>
<tr>
<th>Metric</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>precision</td>
<td>5</td>
<td>0.95652</td>
<td>0.89473</td>
<td>0.84737</td>
<td>1</td>
<td>0.94176 0.93743</td>
</tr>
<tr>
<td>1</td>
<td>recall</td>
<td>5</td>
<td>1.00000</td>
<td>1.00000</td>
<td>1.00000</td>
<td>1</td>
<td>1.00000</td>
</tr>
<tr>
<td>2</td>
<td>F1</td>
<td>5</td>
<td>0.97777</td>
<td>0.94444</td>
<td>0.94444</td>
<td>1</td>
<td>0.968697 0.967273</td>
</tr>
</tbody>
</table>
```
f1 of CV on H(l) by **RF** is 100.0% with both positive and negative rules:

```python
result = mg.cv_matcher_and_trigger(rf, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4, neg_trigger1],
    table = H, exclude_atts=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
    target_attr='gold', random_state = 1280)
result['cv_stats']
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

f1 increased From **0.9835 → 0.9672 → 1.0** along with adding rules

(2)

f1 of CV on H(l) by **NB** is 97.79% without any rules:

```python
result = mg.cv_matcher_and_trigger(nb, {}),
    table = H, exclude_atts=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
    target_attr='gold', random_state = 1280)
result['cv_stats']
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1.000000</td>
<td>1</td>
<td>1.000000</td>
<td>1</td>
<td>1</td>
<td>1.000000</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0.595991</td>
<td>1</td>
<td>0.882353</td>
<td>1</td>
<td>1</td>
<td>0.958289</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0.952381</td>
<td>1</td>
<td>0.937500</td>
<td>1</td>
<td>1</td>
<td>0.977976</td>
</tr>
</tbody>
</table>

f1 of CV on H(l) by **NB** is 97.26% with only positive rules:

```python
result = mg.cv_matcher_and_trigger(nb, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4],
    table = H, exclude_atts=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
    target_attr='gold', random_state = 1280)
result['cv_stats']
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>0.956522</td>
<td>0.944444</td>
<td>0.894737</td>
<td>1</td>
<td>0.941178</td>
<td>0.947375</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0.977778</td>
<td>0.971429</td>
<td>0.944444</td>
<td>1</td>
<td>0.968697</td>
<td>0.972570</td>
</tr>
</tbody>
</table>
f1 of CV on H(I) by NB is 100.0% with both positive and negative rules:

```python
def cv_matcher_end_trigger(nb, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4, neg_trigger1],
    table = H, exclude_attr=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
    target_attr='gold', random_state = 1200)
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>Num folds</th>
<th>Fold 1</th>
<th>Fold 2</th>
<th>Fold 3</th>
<th>Fold 4</th>
<th>Fold 5</th>
<th>Mean score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

f1 increased From 0.9779 → 0.9726 → 1.0 along with adding rules

**important note for adding rules:**

a. I only gave two examples above to show how the rules effect matchers’ accuracy step by step. Basically, the positive rule is in charge of increasing recall. On the other hand, the negative rule is used to improve precision. It makes sense that positive rule assign positive label to the matching pairs once the criteria is met. The True Positive is increasing while the False Positive is increasing.

b. The order of applying rules is:
   
   (pos_trigger1 + pos_trigger2 + pos_trigger3+pos_trigger4 + neg_trigger1)

   the most important reason for adding the negative rule at the end is owing to that the positive rules are very loose compared with negative rule. Thus I need the negative rule to rectify the final result in the end. in other words, negative rule is more strong and precise in our case.

4. **Comparison**

Finally for each of the six learning methods, train the matcher based on that method on I, then report its precision/recall/F-1 on J.
RF:

```python
# Get feature vectors
N = mg.extract_feature_vecs(J, feature_table=feat_subset_all, attr_after='gold')
N.fillna(0, inplace=True)

# Train using feature vectors from I
rf.fit(table=N,
   exclude_attr=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
   target_attr='gold')

# Predict N
N = rf.predict(table=N, exclude_attr=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
   append=True, target_attr='predicted', inplace=False)

# Apply trigger
T1 = pos_trigger1.execute(N, 'predicted', inplace=False)
T2 = pos_trigger2.execute(T1, 'predicted', inplace=False)
T3 = pos_trigger3.execute(T2, 'predicted', inplace=False)
T4 = pos_trigger4.execute(T3, 'predicted', inplace=False)
T5 = neg_trigger.execute(T4, 'predicted', inplace=False)

# Evaluate the result
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)
```

After adding rules: Precision: 100.0% (36/36), Recall: 100.0% (36/36), **F1: 100.0%**
False positives: 0 (out of 36 positive predictions)
False negatives: 0 (out of 84 negative predictions)

DT:

```python
# Get feature vectors
N = mg.extract_feature_vecs(J, feature_table=feat_subset_all, attr_after='gold')
N.fillna(0, inplace=True)

# Train using feature vectors from I
dt.fit(table=N,
   exclude_attr=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
   target_attr='gold')

# Predict N
N = dt.predict(table=N, exclude_attr=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
   append=True, target_attr='predicted', inplace=False)

# Apply trigger
T1 = pos_trigger1.execute(N, 'predicted', inplace=False)
T2 = pos_trigger2.execute(T1, 'predicted', inplace=False)
T3 = pos_trigger3.execute(T2, 'predicted', inplace=False)
T4 = pos_trigger4.execute(T3, 'predicted', inplace=False)
T5 = neg_trigger.execute(T4, 'predicted', inplace=False)

# Evaluate the result
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)
```

After adding rules: Precision: 97.3% (36/37), Recall: 100.0% (36/36), **F1: 98.63%**
False positives: 1 (out of 37 positive predictions)
False negatives: 0 (out of 83 negative predictions)
SVM:

```python
# Get feature vectors
M = mg.extract_feature_vecs(J, feature_table='feat_subset_all', attrs_after='gold')
M.fillna(0, inplace=True)

# Train using feature vectors from I
svm.fit(table=M,
        exclude_attr=['_id', 'itable.record_id', 'rtable.record_id', 'gold'],
        target_attr='gold')

# Predict M
N = svm.predict(table=M, exclude_attr=['_id', 'itable.record_id', 'rtable.record_id', 'gold'],
                append=True, target_attr='predicted', inplace=False)

# Apply trigger
T1 = pos_trigger1.execute(N, 'predicted', inplace=False)
T2 = pos_trigger2.execute(T1, 'predicted', inplace=False)
T3 = pos_trigger3.execute(T2, 'predicted', inplace=False)
T4 = pos_trigger4.execute(T3, 'predicted', inplace=False)
T5 = neg_trigger1.execute(T4, 'predicted', inplace=False)

# Evaluate the result
print('Fraud prediction with rules:
	precision : 97.22\% (35/36)
	recall : 97.22\% (35/36)
	F1 : 97.22\%
	false positives : 1 (out of 36 positive predictions)
	false negatives : 1 (out of 84 negative predictions)
')
```

After adding rules: **Precision : 97.22\% (35/36), Recall : 97.22\% (35/36), F1 : 97.22\%**
False positives : 1 (out of 36 positive predictions)
False negatives : 1 (out of 84 negative predictions)

NB:

```python
# Get feature vectors
M = mg.extract_feature_vecs(J, feature_table='feat_subset_all', attrs_after='gold')
M.fillna(0, inplace=True)

# Train using feature vectors from I
nb.fit(table=M,
       exclude_attr=['_id', 'itable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = nb.predict(table=M, exclude_attr=['_id', 'itable.record_id', 'rtable.record_id', 'gold'],
               append=True, target_attr='predicted', inplace=False)

# Apply trigger
T1 = pos_trigger1.execute(N, 'predicted', inplace=False)
T2 = pos_trigger2.execute(T1, 'predicted', inplace=False)
T3 = pos_trigger3.execute(T2, 'predicted', inplace=False)
T4 = pos_trigger4.execute(T3, 'predicted', inplace=False)
T5 = neg_trigger1.execute(T4, 'predicted', inplace=False)

# Evaluate the result
print('Fraud prediction with rules:
	precision : 100.0\% (36/36)
	recall : 100.0\% (36/36)
	F1 : 100.0\%
	false positives : 0 (out of 36 positive predictions)
	false negatives : 0 (out of 84 negative predictions)
')
```

After adding rules: **Precision : 100.0\% (36/36), Recall : 100.0\% (36/36), F1 : 100.0\%**
False positives : 0 (out of 36 positive predictions)
False negatives : 0 (out of 84 negative predictions)
LN:

```python
# Get feature vectors
M = mg.extract_feature_vecs(J, feature_table='feat_subset_all', attrs_after='gold')
M.fillna(0, inplace=True)

# Train using feature vectors from I
ln.fit(table=M,
       excludeAttrs=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
       targetAttr='gold')

# Predict M
N = ln.predict(table=M, excludeAttrs=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
               append=True, targetAttr='predicted', inplace=False)

# Apply trigger
T1 = pos_trigger1.execute(N, 'predicted', inplace=False)
T2 = pos_trigger2.execute(T1, 'predicted', inplace=False)
T3 = pos_trigger3.execute(T2, 'predicted', inplace=False)
T4 = pos_trigger4.execute(T3, 'predicted', inplace=False)
T5 = neg_trigger1.execute(T4, 'predicted', inplace=False)

# Evaluate the result
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

# Without triggers:
# Precision : 97.34 (36/37)
# Recall : 100.0% (36/36)
# F1 : 98.63%
# False positives : 1 (out of 37 positive predictions)
# False negatives : 0 (out of 36 negative predictions)
```

After adding rules: Precision: 97.3% (36/37), Recall: 100.0% (36/36), \( F_1: 98.63\% \)
False positives: 1 (out of 37 positive predictions)
False negatives: 0 (out of 36 negative predictions)

LG:

```python
# Get feature vectors
M = mg.extract_feature_vecs(J, feature_table='feat_subset_all', attrs_after='gold')
M.fillna(0, inplace=True)

# Train using feature vectors from I
lg.fit(table=M,
       excludeAttrs=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
       targetAttr='gold')

# Predict M
N = lg.predict(table=M, excludeAttrs=['_id', 'ltable.record_id', 'rttable.record_id', 'gold'],
               append=True, targetAttr='predicted', inplace=False)

# Apply trigger
T1 = pos_trigger1.execute(N, 'predicted', inplace=False)
T2 = pos_trigger2.execute(T1, 'predicted', inplace=False)
T3 = pos_trigger3.execute(T2, 'predicted', inplace=False)
T4 = pos_trigger4.execute(T3, 'predicted', inplace=False)
T5 = neg_trigger1.execute(T4, 'predicted', inplace=False)

# Evaluate the result
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

# Without triggers:
# Precision : 97.34 (36/37)
# Recall : 100.0% (36/36)
# F1 : 98.63%
# False positives : 1 (out of 37 positive predictions)
# False negatives : 0 (out of 36 negative predictions)
```

After adding rules: Precision: 100.0% (36/36), Recall: 100.0% (36/36), \( F_1: 100.0\% \)
False positives: 0 (out of 36 positive predictions)
False negatives: 0 (out of 84 negative predictions)
For the final best learning method Y selected, train it on I, then report its precision/recall/F-1 on J. The Y is RF without rules as shown above. Its prediction on J is:

#Precision : 94.59% (35/37)
#Recall : 97.22% (35/36)
#F1 : 95.89%
#False positives : 2 (out of 37 positive predictions)
#False negatives : 1 (out of 83 negative predictions)

For the final best matcher (that is, Y*, which is the learning-based method Y plus the rules), train it on I then report its precision/recall/F-1 on J. Its prediction on J is:

#Precision : 100.0% (36/36)
#Recall : 100.0% (36/36)
#F1 : 100.0%
#False positives : 0 (out of 36 positive predictions)
#False negatives : 0 (out of 84 negative predictions)

5. Misc

a. More than 3 hours for labeling and relabeling the data. label_table method in Magellan is very convenient to label data. However, it’s not friendly to be used for relabeling data.

b. Approximately 7 hours are spent to find the best learning matcher.

c. More than 50 hours are spent to play around adding rules and improvement.