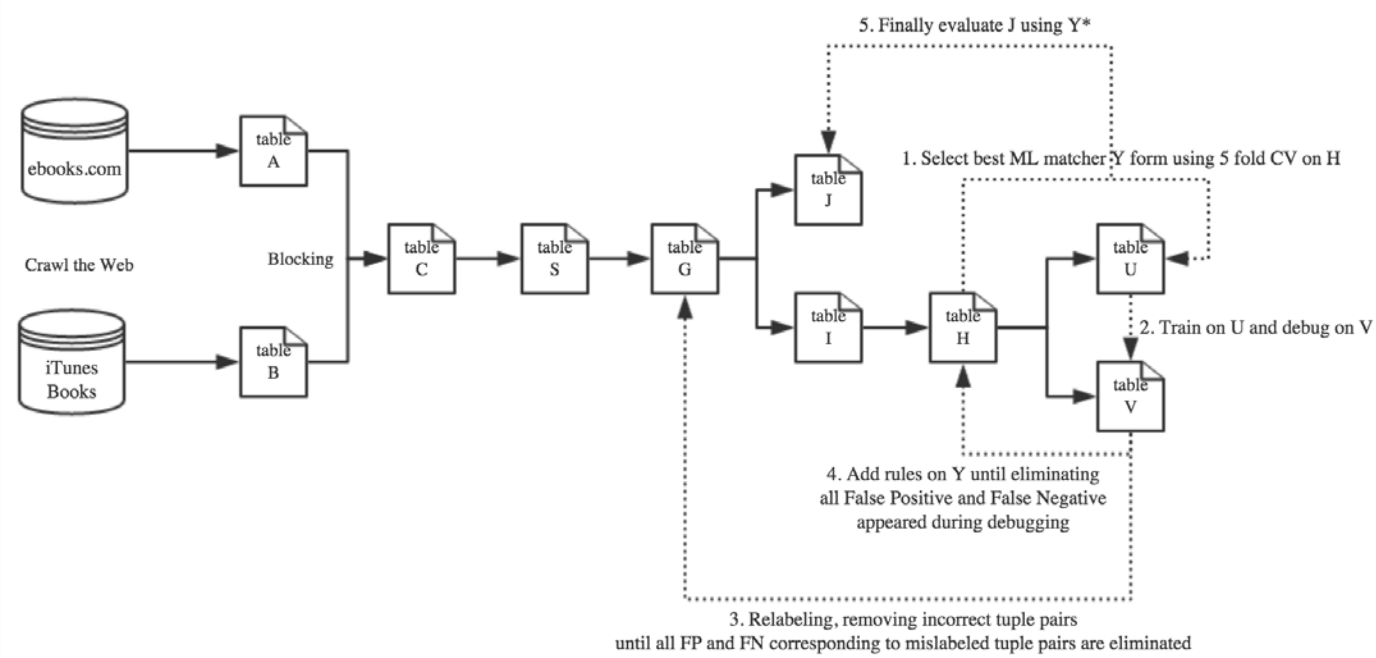
CS 784 DATA MODELS PROJECT

Department of Computer Science
University of Wisconsin – Madison

Shuang Wu

Table of Contents

1. Workflow



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**

.
.
.

 # 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

.
.
.

<u>important note for Debugging Iteration 2:</u>

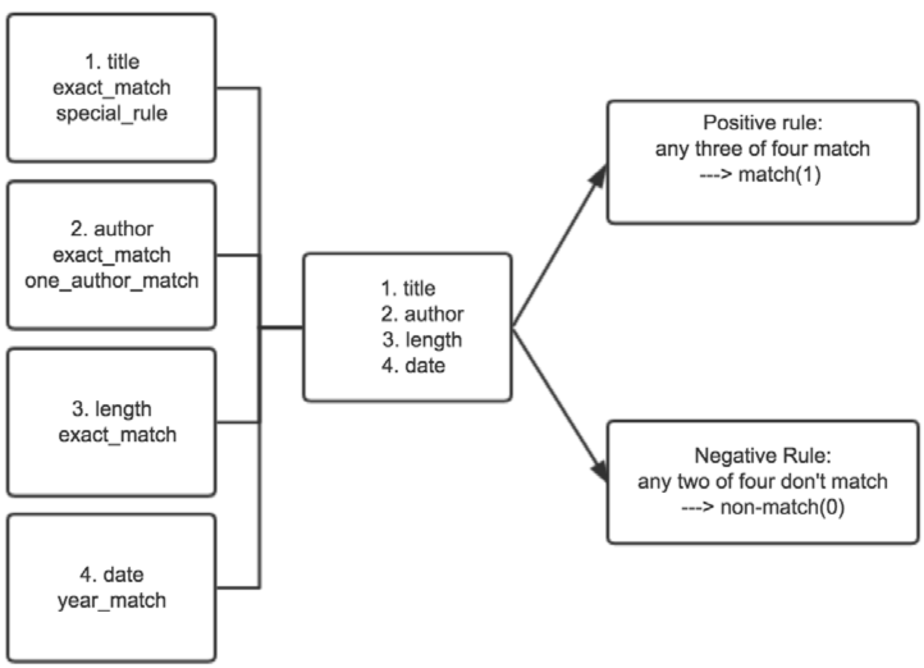
a. **The most confusing part in this matching scenario is, many books (with the same title) might have many different versions. For those cases, I treat different versions of the book as different books. As for some very ambiguous pairs, I went a third party (Amazon and noble & barnes) to check if they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

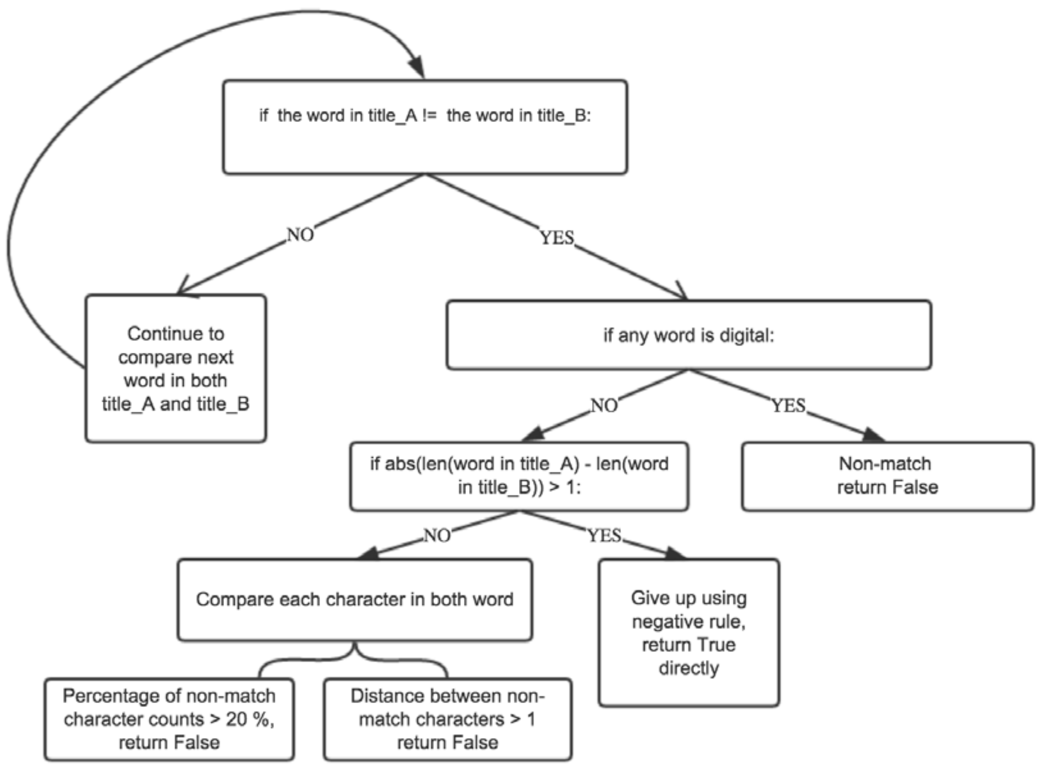
   case 1 - Messi 2016 Updated Edition VS. Messi 2014 Updated Edition
   case 2 - Golfâs Finest Par Threes VS. Golfs Finest Par Threes
   case 3 - Suarez â 2016 Updated Edition VS. Ronaldo â 2016 Updated Edition

   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
   <u><not match if 'title_title_jac_qgm_3_qgm_3(ltuple, rtuple) < 0.9' is true></u>
   to improve accuracy.

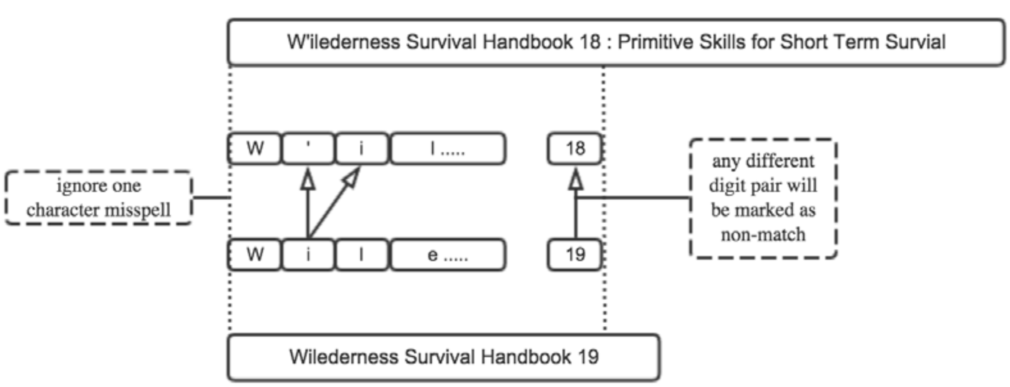
c. **Based on observations from a and b above, I designed a more logical rule to make decision. The basic strategy is:**



Special rule for title:



More specifically for a special case (title):



Finally this trigger will solve some issues like those:

Left Tuple

	Value
record_id	1818
publisher	Icon Books
date	Aug 06, 2015
description	Luis SuÃ¡rez is one of the most brilliant and controversial players in world football.Signed by Ba…
language	English
title	Suarez â 2016 Updated Edition
url	https://itunes.apple.com/us/book/suarez-2016-updated-edition/id988699203?mt=11
rating_value	nan
price	7.99
author	Luca Caioli
rating_star	0.0
seller	Directebooks Ltd
short_description	The Extraordinary Story Behind Football's Most Explosive Talent
length	240
genre	Soccer
page_id	988699203

Right Tuple

	Value
record_id	11057
ISBN	9.78190685094e+12
description	Season after season, Cristiano Ronaldo continues to prove that he is one of footballâs true greats. A thre…
price	7.99
date	August 2015
publisher	Icon Books
review_count	nan
title	Ronaldo â 2016 Updated Edition
rating_value	nan
author	Luca Caioli
length	240
short_description	The Obsession For Perfection

Left Tuple

	Value
record_id	909
publisher	Tuttle Publishing
date	Aug 07, 2012
description	The illustrations are clear and the instructions are simple, and a reasonabl…
language	English
title	Practical Karate Volume 4
url	https://itunes.apple.com/us/book/practical-karate-volume-4/id957275633?mt=11
rating_value	nan
price	6.99
author	Donn F. Draeger & Masatoshi Nakayama
rating_star	0.0
seller	The Perseus Books Group, LLC
short_description	Defense Against Armed Assailants
length	122
genre	Sports & Outdoors
page_id	957275633

Right Tuple

	Value
record_id	6508
ISBN	9.78146290516e+12
description	"Simple, clear, easy to learnâ¦Dispenses with hours of needed to practice for the …
price	9.95
date	July 2012
publisher	Tuttle Publishing
review_count	nan
title	Practical Karate volume 1
rating_value	nan
author	Donn F. Draeger, Masatoshi Nakayama
length	112
short_description	Fundamentals of Self-Defense

Left Tuple

	Value
record_id	2385
publisher	McGraw-Hill Education
date	Apr 16, 2010
description	An essential guide to everything you need to stay sheltered, fed, healthy, and safe in the backcountry…
language	English
title	Wilderness Survival Handbook : Primitive Skills for Short-Term Survival and Long-Term Comfort
url	https://itunes.apple.com/us/book/wilderness-survival-handbook/id498457182?mt=11
rating_value	nan
price	14.99
author	Michael Pewtherer
rating_star	0.0
seller	The McGraw-Hill Companies, Inc.
short_description	nan
length	288
genre	Outdoors
page_id	498457182

Right Tuple

	Value
record_id	10067
ISBN	9.78007178268e+12
description	Do you have what it takes?Youâre alone in the wilderness with nothing but a knife and the clothes …
price	18.0
date	May 2006
publisher	McGraw-Hill Education
review_count	nan
title	Wilderness Survival
rating_value	nan
author	Mark Elbroch, Michael Pewtherer
length	288
short_description	Living Off the Land with the Clothes on Your Back and the Knife on Your Belt

Left Tuple

	Value
publisher	Vertebrate Publishing
date	Jul 01, 2014
description	To those who went to the War straight from school and survived it, the problem of wha…
language	English
title	Snow on the Equator
url	https://itunes.apple.com/us/book/snow-on-the-equator/id896714427?mt=11
rating_value	nan
price	5.99
author	H.W. Tilman & Jim Perrin
rating_star	0.0
seller	The Perseus Books Group, LLC
short_description	nan
length	300
genre	Mountaineering
page_id	896714427

Right Tuple

	Value
record_id	1592
ISBN	9.78190946115e+12
description	To those who went to the War straight from school and survived it, the problem of what t…
price	17.5
date	September 2015
publisher	Vertebrate Publishing
review_count	nan
title	Snow on the Equator
rating_value	nan
author	H.W. Tilman, Sir Chris Bonington
length	300
short_description	Mount Kenya, Kilimanjaro and the great African odyssey

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

precision:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<magellan.matcher.dtmatcher.DTMatcher object a...	5	0.916667	0.875000	0.937500	0.809524	1.000000	0.907738
1	RF	<magellan.matcher.rfmatcher.RFMatcher object a...	5	0.916667	0.933333	0.933333	0.947368	1.000000	0.946140
2	SVM	<magellan.matcher.svmmatcher.SVMMatcher object...	5	0.840000	0.882353	0.823529	0.900000	1.000000	0.889176
3	NB	<magellan.matcher.nbmatcher.NBMatcher object a...	5	0.880000	0.937500	0.882353	0.894737	1.000000	0.918918
4	LogReg	<magellan.matcher.logregmatcher.LogRegMatcher ...	5	0.913043	0.937500	0.823529	0.900000	1.000000	0.914815
5	LinReg	<magellan.matcher.linregmatcher.LinRegMatcher ...	5	0.840000	0.882353	0.937500	0.947368	0.933333	0.908111

recall:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<magellan.matcher.dtmatcher.DTMatcher object a...	5	1.000000	0.823529	0.9375	0.944444	0.928571	0.926809
1	RF	<magellan.matcher.rfmatcher.RFMatcher object a...	5	1.000000	0.823529	0.8750	1.000000	0.928571	0.925420
2	SVM	<magellan.matcher.svmmatcher.SVMMatcher object...	5	0.954545	0.882353	0.8750	1.000000	1.000000	0.942380
3	NB	<magellan.matcher.nbmatcher.NBMatcher object a...	5	1.000000	0.882353	0.9375	0.944444	1.000000	0.952859
4	LogReg	<magellan.matcher.logregmatcher.LogRegMatcher ...	5	0.954545	0.882353	0.8750	1.000000	1.000000	0.942380
5	LinReg	<magellan.matcher.linregmatcher.LinRegMatcher ...	5	0.954545	0.882353	0.9375	1.000000	1.000000	0.954880

f1:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<magellan.matcher.dtmatcher.DTMatcher object a...	5	0.956522	0.848485	0.937500	0.871795	0.962963	0.915453
1	RF	<magellan.matcher.rfmatcher.RFMatcher object a...	5	0.956522	0.875000	0.903226	0.972973	0.962963	0.934137
2	SVM	<magellan.matcher.svmmatcher.SVMMatcher object...	5	0.893617	0.882353	0.848485	0.947368	1.000000	0.914365
3	NB	<magellan.matcher.nbmatcher.NBMatcher object a...	5	0.936170	0.909091	0.909091	0.918919	1.000000	0.934654
4	LogReg	<magellan.matcher.logregmatcher.LogRegMatcher ...	5	0.933333	0.909091	0.848485	0.947368	1.000000	0.927656
5	LinReg	<magellan.matcher.linregmatcher.LinRegMatcher ...	5	0.893617	0.882353	0.937500	0.972973	0.965517	0.930392

b. After the first time CV, RF is chosen as best classifier because it has highest accuracy of 94.61%

<magellan.matcher.rfmatcher.RFMatcher object a...	5	0.916667	0.933333	0.933333	0.947368	1.000000	0.946140

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:

Left Tuple

	Value
record_id	2973
publisher	Falcon Guides
date	Feb 01, 2011
description	Backpacker's Using a GPS: Digital Trip Planning, Recording, and Sharing is a complete guide to ...
language	English
title	Backpacker Magazine's Using a GPS
url	https://itunes.apple.com/us/book/backpacker-magazines-using/id938494836?mt=11
rating_value	nan
price	11.99
author	Bruce Grubbs
rating_star	0.0
seller	The Rowman & Littlefield Publishing Group
short_description	Digital Trip Planning, Recording, And Sharing
length	96
genre	Outdoors
page_id	938494836

Right Tuple

	Value
record_id	5851
ISBN	9.78076276778e+12
description	Backpacker magazineâs Outdoor Knots brings you essential mind gear from the two most res...
price	11.99
date	February 2011
publisher	Falcon Guides
review_count	nan
title	Backpacker Magazine's Outdoor Knots
rating_value	nan
author	Clyde Soles
length	96
short_description	The Knots You Need To Know

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:

Left Tuple

	Value
record_id	177
publisher	Triumph Books
date	Mar 01, 2010
description	162-0: Imagine a Twins Perfect Season imagines that season by identifying the mo…
language	English
title	162-0: Imagine a Twins Perfect Season
url	https://itunes.apple.com/us/book/162-0-imagine-twins-perfect/id708499380?mt=11
rating_value	nan
price	11.99
author	Dave Wright
rating_star	0.0
seller	Chicago Review Press, Inc. DBA Independent Publishers Group
short_description	The Greatest Wins!
length	304
genre	Baseball
page_id	708499380

Right Tuple

	Value
record_id	6452
ISBN	9.78161749074e+12
description	162-0: Imagine a Red Sox Perfect Season imagines that season by identifying the m…
price	11.99
date	March 2010
publisher	Triumph Books
review_count	nan
title	162-0: Imagine a Red Sox Perfect Season
rating_value	nan
author	Mark Cofman, Tony Massaroti
length	304
short_description	The Greatest Wins!

split on G with random_state = 50 and split on H with random_state = 0, after 2nd round of cleaning, we get

 # 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:

Left Tuple

	Value
record_id	6480
publisher	Charlesbridge
date	Sep 01, 2009
description	This book hits a grand slam right out of the park! No diehard devotee of the ...
language	English
title	Book of Baseball Stuff
url	https://itunes.apple.com/us/book/book-of-baseball-stuff/id801564884?mt=11
rating_value	nan
price	8.99
author	Ron Martriano
rating_star	0.0
seller	Random House, LLC
short_description	nan
length	192
genre	Baseball
page_id	801564884

Right Tuple

	Value
record_id	8598
ISBN	9.78160734509e+12
description	Touchdown! These tales from the gridiron will set fans abuzz. Fun, fille...
price	8.99
date	July 2011
publisher	Charlesbridge
review_count	nan
title	Book of Football Stuff
rating_value	nan
author	Ron Martirano
length	192
short_description	nan

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).

precision:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<magellan.matcher.dtmatcher.DTMatcher object a...	5	0.947368	1.000000	0.952381	1.0	1	0.979950
1	RF	<magellan.matcher.rfmatcher.RFMatcher object a...	5	1.000000	1.000000	1.000000	1.0	1	1.000000
2	SVM	<magellan.matcher.svmmatcher.SVMMatcher object...	5	1.000000	1.000000	0.833333	1.0	1	0.966667
3	NB	<magellan.matcher.nbmatcher.NBMatcher object a...	5	1.000000	1.000000	1.000000	1.0	1	1.000000
4	LogReg	<magellan.matcher.logregmatcher.LogRegMatcher ...	5	1.000000	1.000000	0.952381	0.9	1	0.970476
5	LinReg	<magellan.matcher.linregmatcher.LinRegMatcher ...	5	0.900000	0.916667	1.000000	1.0	1	0.963333

recall:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<magellan.matcher.dtmatcher.DTMatcher object a...	5	1	1.000000	1.00	1	0.894737	0.978947
1	RF	<magellan.matcher.rfmatcher.RFMatcher object a...	5	1	1.000000	1.00	1	0.894737	0.978947
2	SVM	<magellan.matcher.svmmatcher.SVMMatcher object...	5	1	0.909091	1.00	1	0.947368	0.971292
3	NB	<magellan.matcher.nbmatcher.NBMatcher object a...	5	1	0.909091	0.95	1	0.842105	0.940239
4	LogReg	<magellan.matcher.logregmatcher.LogRegMatcher ...	5	1	0.909091	1.00	1	0.947368	0.971292
5	LinReg	<magellan.matcher.linregmatcher.LinRegMatcher ...	5	1	1.000000	1.00	1	0.947368	0.989474

f1:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<magellan.matcher.dtmatcher.DTMatcher object a...	5	0.972973	1.000000	0.975610	1.000000	0.944444	0.978605
1	RF	<magellan.matcher.rfmatcher.RFMatcher object a...	5	1.000000	1.000000	1.000000	1.000000	0.944444	0.988889
2	SVM	<magellan.matcher.svmmatcher.SVMMatcher object...	5	1.000000	0.952381	0.909091	1.000000	0.972973	0.966889
3	NB	<magellan.matcher.nbmatcher.NBMatcher object a...	5	1.000000	0.952381	0.974359	1.000000	0.914286	0.968205
4	LogReg	<magellan.matcher.logregmatcher.LogRegMatcher ...	5	1.000000	0.952381	0.975610	0.947368	0.972973	0.969666
5	LinReg	<magellan.matcher.linregmatcher.LinRegMatcher ...	5	0.947368	0.956522	1.000000	1.000000	0.972973	0.975373

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 :

```
result = mg.cv_matcher_and_trigger(rf, [], table = H,
                                   exclude_attrs=['_id', 'ltable.id', 'rtable.id', 'gold'],
                                  target_attr='gold',random_state = 1200)

result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 0.335 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	1.000000	0.944444	1.000000	1	1	0.988889
1	recall	5	0.954545	1.000000	0.941176	1	1	0.979144
2	f1	5	0.976744	0.971429	0.969697	1	1	0.983574

I add short_description and delete price, and then extract feature vectors.

```
# 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[47:54]

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

```
result = mg.cv_matcher_and_trigger(rf, [], table = H,
                                   exclude_attrs=['_id', 'ltable.id', 'rtable.id', 'gold'],
                                  target_attr='gold',random_state = 1200)

result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 0.349 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	1	0.944444	1.000000	1	1	0.988889
1	recall	5	1	1.000000	0.941176	1	1	0.988235
2	f1	5	1	0.971429	0.969697	1	1	0.988225

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:

```
feature1 = mg.get_feature_fn("exact_match(ltuple['title'], rtuple['title'])", mg._match_t, mg._match_s)
mg.add_feature(feat_table, 'title_title_exm', feature1)

feature2 = mg.get_feature_fn("exact_match(ltuple['author'], rtuple['author'])", mg._match_t, mg._match_s)
mg.add_feature(feat_table, 'author_author_exm', feature2)
```

add year_match:

```
 # x, y will be of type pandas series  e.g.   19-march-15  &  march-15
def match_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:

```
 # x, y will be of type pandas series  e.g.   19-march-15  &  march-15import re
def match_one_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 = y_dateSet
        large = x_dateSet
    else:
        small = x_dateSet
        large = y_dateSet

    small_size = len(small)
    large_size = len(large)

    for index1 in range (0, small_size):
        for index2 in range(0, large_size):
            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:

```
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 aviod 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 thier 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 & volume 102
        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. Golfâs & Golfs
        # 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_shorter_String
                shorter_word = word_in_longer_String
            else:
                shorter_word = word_in_shorter_String
                longer_word = word_in_longer_String

            count = 0  # count for # of letter no 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. Gâolf & 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
                        pointer2 += 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 Rule1 to Rule4:

```
# 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_exm(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 = H, exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
                                  target_attr='gold',random_state = 1200)
result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 0.250 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	1.000000	0.944444	1.000000	1	1	0.988889
1	recall	5	0.954545	1.000000	0.941176	1	1	0.979144
2	f1	5	0.976744	0.971429	0.969697	1	1	0.983574

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 = H, exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
                                  target_attr='gold',random_state = 1200)
result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 2.252 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	0.956522	0.894737	0.894737	1	0.941176	0.937434
1	recall	5	1.000000	1.000000	1.000000	1	1.000000	1.000000
2	f1	5	0.977778	0.944444	0.944444	1	0.969697	0.967273

f1 of CV on H(I) by RF is 100.0% with both positive and negative rules:

```
result = mg.cv_matcher_and_trigger(rf, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4, neg_trigger1],
                                   table = H, exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
                                  target_attr='gold',random_state = 1200)
result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 2.562 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	1	1	1	1	1	1
1	recall	5	1	1	1	1	1	1
2	f1	5	1	1	1	1	1	1

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

(2)

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

```
result = mg.cv_matcher_and_trigger(nb, [],
                                   table = H, exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
                                  target_attr='gold',random_state = 1200)
result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 0.230 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	1.000000	1	1.000000	1	1	1.000000
1	recall	5	0.909091	1	0.882353	1	1	0.958289
2	f1	5	0.952381	1	0.937500	1	1	0.977976

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

```
result = mg.cv_matcher_and_trigger(nb, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4],
                                   table = H, exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
                                  target_attr='gold',random_state = 1200)
result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 2.536 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	0.956522	0.944444	0.894737	1	0.941176	0.947376
1	recall	5	1.000000	1.000000	1.000000	1	1.000000	1.000000
2	f1	5	0.977778	0.971429	0.944444	1	0.969697	0.972670

f1 of CV on H(I) by NB is 100.0% with both positive and negative rules:

```
result = mg.cv_matcher_and_trigger(nb, [pos_trigger1, pos_trigger2, pos_trigger3, pos_trigger4, neg_trigger1],
                                    table = H, exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
                                   target_attr='gold',random_state = 1200)
result['cv_stats']
```

```
0%  100%
[#####] | ETA[sec]: 0.000
Total time elapsed: 2.607 sec
```

	Metric	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	precision	5	1	1	1	1	1	1
1	recall	5	1	1	1	1	1	1
2	f1	5	1	1	1	1	1	1

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

<u>important note for adding rules:</u>

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:

```
# 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
rf.fit(table= H,
       exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = rf.predict(table=M, exclude_attrs=['_id', 'ltable.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
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

#without triggers:
#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)
```

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:

```
# 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
dt.fit(table= H,
       exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = dt.predict(table=M, exclude_attrs=['_id', 'ltable.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
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

#without triggers:
#Precision : 92.11% (35/38)
#Recall : 97.22% (35/36)
#F1 : 94.59%
#False positives : 3 (out of 38 positive predictions)
#False negatives : 1 (out of 82 negative predictions)
```

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:

```
# 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= H,
       exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = svm.predict(table=M, exclude_attrs=['_id', 'ltable.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
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

#without triggers:
#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:

```
# 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= H,
       exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = nb.predict(table=M, exclude_attrs=['_id', 'ltable.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
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

#without triggers:
#Precision : 100.0% (35/35)
#Recall : 97.22% (35/36)
#F1 : 98.59%
#False positives : 0 (out of 35 positive predictions)
#False negatives : 1 (out of 85 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:

```
# 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= H,
       exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = ln.predict(table=M, exclude_attrs=['_id', 'ltable.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
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

#without triggers:
#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)
```

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)

LG:

```
# 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= H,
       exclude_attrs=['_id', 'ltable.record_id', 'rtable.record_id', 'gold'],
       target_attr='gold')

# Predict M
N = lg.predict(table=M, exclude_attrs=['_id', 'ltable.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
eval_result = mg.eval_matches(T5, 'gold', 'predicted')
mg.print_eval_summary(eval_result)

#without triggers:
#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)
```

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)

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.