REPORT CS784

Priyanka Nayek
Praneeth Kumar Naramsetti

In the project of course-CS784 we worked on a system named Magellan. Magellan is an Entity Matching management system.
Till now we have crawled on two websites: Amazon.com and Barnesandnobles.com and extracted two tables (AMAZON.csv, BARNES.csv) from these websites. We then blocked the two tables to obtain another table-TableC which is a candidate set table obtained from AMAZON.csv(A) and BARNES.csv(B) after blocking. In this report we have documented the entire process (after blocking) of matching the two tables A & B using Magellan.

We started with loading the two tables (A & B) and the blocked table TableC (that we obtained in stage 2 of the project) in memory. We then sampled 450 tuple pairs from tableC. Using these 450 tuple pairs we will test and select the matcher that we will use for matching the two tables A & B. We labeled these 450 tuple pairs to create golden data. We named the golden table G. We will use this G in our matching process with Magellan. We then splitted G into a development set I of 337 examples and an evaluation set J of 113 examples. In our development stage we used I to find the best matcher and then in the production stage we used this best matcher to report its precision and recall on J.

In the next step we converted each tuple pair of I into a feature vector using Magellan. We used Magellan to create this set of features automatically for us. We used a subset of features from the feature set created by Magellan. We can use as many features as possible from the feature set to create the feature vector table but we started with a subset of features of Title and Author attributes. We named this feature vector table (of I) as K. In this feature vector table each tuple is a feature vector together with a golden label. The matchers that currently Magellan supports are:

- Decision Tree
- Random Forest
- Support Vector Machine
- Naive Bayes
- Logistic Regression
- Linear Regression

To decide which matcher method to use we cross-validated all six matcher methods. For the first time we used features on Title and Author attributes. Using these features the cross-validation result is as shown in Table1:

	Name	**f1**	**Precision**	**Recall**
0	DecisionTree	0.900181	0.907251	0.901194
1	RF	0.916124	1.000000	0.848655
2	SVM_06794146326573248441	0.841085	0.896484	0.808767
3	NB	0.773246	0.684441	0.898655
4	LogReg	0.868132	0.962103	0.797656
5	LinReg	0.847050	0.833244	0.866184

Table1: cross-validation result on title and author features

In this cross-validation we observed that Random Forest (RF) gave good precision and f-1 mean score. As RF gave good accuracy so we selected RF as a learning based matcher after this cross-validation. We

considered only precision and f-1 for this selection and not recall because they are more important measures to validate a matcher.
We then split K into U and V to debug RF and increase its recall. We first trained RF using U and tested RF using V. The prediction accuracy that we got on V is as below:

Precision : 100.0% (25/25)
Recall : 89.29% (25/28)
F1 : 94.34%
False positives : 0 (out of 25 positive predictions)
False negatives : 3 (out of 110 negative predictions)

We tried to improve the accuracy. We observed that Levenshtein is decreasing the accuracy as there are tuple pairs which are matching but have word differences. In such cases we know that Levenshtein is not a good similarity function. E.g., Microsoft Office 10, Edition 3 and Microsoft Office 10.
Now, the accuracy that we got is:

Precision : 100.0% (26/26)
Recall : 92.86% (26/28)
F1 : 96.3%
False positives : 0 (out of 26 positive predictions)
False negatives : 2 (out of 109 negative predictions)
Removing levenshtein feature vectors from the feature vector subset increased the precision.
We cross validated again to see if RF is performing better. We see the cross-validation result in the below table2:

	Name	**f1**	**Precision**	**Recall**
0	DecisionTree	0.903804	0.891378	0.923416
1	RF	0.948206	1.000000	0.901779
2	SVM_06794146326573248441	0.832383	0.823099	0.848002
3	NB	0.742698	0.644276	0.886891
4	LogReg	0.868132	0.962103	0.797656
5	LinReg	0.826786	0.829549	0.837475

Table2: cross-validation result on removing levenshtein features of title and author

We saw that still RF performed better than the other decision methods and is giving better result than the other matcher methods.

Next, we tried to improve the recall more and to do that we first checked the labels. There was an incorrect labeling, so we corrected that.
E.g.,

1169 - The New Digital Age: Reshaping the Future of People, Nations and Business - Eric Schmidt# Jared Cohen
900 - The New Digital Age: Transforming Nations, Businesses, and Our Lives - Eric Schmidt# Jared Cohen
These two books were labeled as matching but they were two non-matching books. So, we corrected the label by changing the label from 1 to 0.

We tried to improve/debug RF more by adding Publisher features. As we know that increasing feature will help the model to learn more. So, we tried to improve our matcher by proving it more features. On adding Publisher features we got the below mentioned accuracy on predicting on V

Precision : 100.0% (25/25)
Recall : 89.29% (25/28)
F1 : 94.34%
False positives : 0 (out of 25 positive predictions)
False negatives : 3 (out of 110 negative predictions)

The cross-validation result after this is:

	Name	**f1**	**Precision**	**Recall**
0	DecisionTree	0.897866	0.889474	0.912305
1	RF	0.920685	0.963235	0.883597
2	SVM_06794146326573248441	0.793673	0.890171	0.736432
3	NB	0.762108	0.687934	0.865322
4	LogReg	0.813676	0.915038	0.740140
5	LinReg	0.850053	0.849599	0.856311

We noticed that this lowered our accuracy (recall). We observed the records and saw that in the Publishers there are word differences. So, we decided to remove Levenshtein similarity function from the Publisher. Moreover, we also noticed that there were records which were matching records having same ISBN-13 and were still getting removed:
E.g.,
Hackers & Painters: Big Ideas from the Computer Age - Paul Graham - O'Reilly Media - 978-1449389550
Hackers and Painters: Big Ideas from the Computer Age - Paul Graham - O'Reilly Media, Incorporated - 978-1449389550

We can see this has everything same only publisher is missing a word.
Now we got,

Precision : 100.0% (26/26)
Recall : 92.86% (26/28)
F1 : 96.3%
False positives : 0 (out of 26 positive predictions)
False negatives : 2 (out of 109 negative predictions)

This improved the recall.
Our cross validation result on V is:

	Name	f1	Precision	Recall
0	DecisionTree	0.895617	0.878651	0.923416
1	RF	0.948225	1.000000	0.902363
2	SVM_06794146326573248441	0.800396	0.887500	0.743011
3	NB	0.764581	0.684825	0.876433
4	LogReg	0.813676	0.915038	0.740140
5	LinReg	0.854759	0.861266	0.856311

RF is still performing better than the other matcher methods.

We tried to improve recall more by adding triggers/rules. We added a rule on ISBN-13. The rule that we added is:
'ISBN_13_ISBN_13_lev(ltuple, rtuple) > 0.95

We added this rule because we saw that such matching records may also get removed if there is no rule for them. To not remove such matching records we added this rule on ISBN-13.

When we cross validated on Matcher + trigger we got good accuracy.

	Metric	Num folds	Mean score
0	precision	5	0.951282
1	recall	5	0.948252
2	f1	5	0.949147

The final evaluation that we got on V using the RF matcher with the features and the trigger is as below:

Precision : 100.0% (26/26)
Recall : 92.86% (26/28)
F1 : 96.3%
False positives : 0 (out of 26 positive predictions)
False negatives : 2 (out of 109 negative predictions)

So, we decided to stop here and use RF matcher and the trigger to test our test data J.

We now finally used this matcher and trigger to test/predict our test split J. The final accuracy that we received on the test data J is as shown below:

Precision : 100.0% (24/24)
Recall : 92.31% (24/26)
F1 : 96.0%
False positives : 0 (out of 24 positive predictions)
False negatives : 2 (out of 89 negative predictions)

So, magellan gave us the above accuracy on a small sample of 450 records.

We cross verified our matcher by testing each of the six learning methods. We trained the matchers based on that matcher method on I, then reported its accuracy on J.
The results that we got on the six learning methods are as shown below:

#Decision Tree
Precision : 96.0% (24/25)
Recall : 92.31% (24/26)
F1 : 94.12%
False positives : 1 (out of 25 positive predictions)
False negatives : 2 (out of 88 negative predictions)

#SVM
Precision : 86.36% (19/22)
Recall : 73.08% (19/26)
F1 : 79.17%
False positives : 3 (out of 22 positive predictions)
False negatives : 7 (out of 91 negative predictions)

#Naive Bayes
Precision : 69.44% (25/36)
Recall : 96.15% (25/26)
F1 : 80.65%
False positives : 11 (out of 36 positive predictions)
False negatives : 1 (out of 77 negative predictions)

#Logistic Regression
Precision : 90.0% (18/20)
Recall : 69.23% (18/26)
F1 : 78.26%
False positives : 2 (out of 20 positive predictions)
False negatives : 8 (out of 93 negative predictions)

#Linear Regression
Precision : 85.19% (23/27)
Recall : 88.46% (23/26)
F1 : 86.79%
False positives : 4 (out of 27 positive predictions)
False negatives : 3 (out of 86 negative predictions)

From all the above results we can confirm and say that the matcher method that we selected for matching the two tables is perfectly best. It performed better than all the other matcher methods.

TIME REQUIRED TO PERFORM THE ENTIRE PROCESS:
==
An approximate time estimate:
(a) how much did it take to label the data
Labeling the 450 tuple pairs was made not difficult (easy) by using Magellan. It took 40mins to properly label 45 samples.
(b) to find the best learning-based matcher
Finding the best learning-based matcher took us approx 7 hrs.
(c) to add rules to the learning-based matcher
Coming up with a good rule and adding rules to the learning-based matcher took us around 2-3 hrs.
Total time required for the entire matching process after Blocking is around 12-15 hrs

FEEDBACKS:
============
Overall working experience with Magellan was good. But, would still like to point out few things.

1. While working with Magellan it used to be very difficult for us to debug the errors that we used to get. There were certain errors which were very general and straightforward and those were easy to debug. But there were certain errors which were thrown by other Python packages imported for Magellan and these errors were really head-breaking to debug. E.g.,
   IndexError: index 9971 is out of bounds for axis 0 with size 9958
   It was very difficult for us to debug this error. But with Pradap's help we solved this issue.
   It would be great if Magellan would be able catch and throw all type of errors.