Entity Matching

Sowrabha Horatti Gopal
sgopal3@wisc.edu

Pradeep Kashyap Ramaswamy
pradeep.kashyap@wisc.edu

Sharath Prabhudeva Hiremath
shiremath@wisc.edu

1 Entity

Our entity is a tuple containing **Book Details**. We have scraped book details from the following two data sources -

1. **Goodreads**
2. **Barnes & Noble**

We chose these data sources as they give us comprehensive details of all the books. Following table gives the schema of entities scraped from **Goodreads** and **Barnes & Noble**

Goodreads	**Barnes & Noble**
Publication	Publisher
ISBN13	ISBN-13
Author	Author
Original_Title	Original_Title
Published_Date	Publication date
Pages	Pages
Ratings	–
Genres	–
Edition_Language	–
ISBN	–
Title	–
Reviews	–
Average_Rating	–
Edition	–

Table 1: Entity Schema of Goodreads and Barnes & Noble

- Number of tuples present in **Goodreads** table - **4,907**
- Number of tuples present in **Barnes & Noble** table - **3,000**

Final Schema being used -

Attributes
Original_Title
Author
ISBN13
Publication
Published_Date

Table 2: Final Schema

2 Blocking

We used three types of blockers for blocking. Following are the details -

2.1 Attribute Equivalence Blocker (AEB)

We performed Attribute Equivalence blocking on **source tables** on the following attributes -

1. **ISBN13:** Book entity has an attribute 'ISBN13' which is a unique code for a book. We performed blocking on this attribute first, which removes the most obvious mismatches. Sometimes it so happens that same book might have different ISBN because of different editions. That's why we continue blocking with other attributes.

2. **Original_title:** Book's title is generally unique for every book and hence we are eliminating candidates which don't match with Title. But, due to the way websites list the book's names, they might not be exact match. E.g., "**Harry Potter and the Deathly Hallows (Harry Potter Series #7)**" and "**Harry Potter and the Deathly Hallows**". In this example, the series name is also included as part of the book name. Another example is "**Slave: My True Story**" and "**Slave : My True Story**". Note in this example, that they look like exact match, but there is an extra space before **':'**. Because of such book names, we continue blocking on other attributes.

3. **Author:** For a candidate tuple to match, the author names should be an exact match. But this might not be true because the author names might be abbreviated. E.g., **J.K. Rowling** and **Joanne K. Rowling**. Similar to book titles, single spaces might make the names to be non matching. E.g., **J.K. Rowling**. As a result we had to continue blocking further.r

Among 4907 and 3000 records of Goodreads and Barnes & Noble respectively, it is expected to have around 1500 matches (because scraped from top sellers list). Equivalence Blocker (AEB) on ISBN resulted in only **347** tuples; Clearly the this is not a good number (when compared to 1500) because of the same books having different ISBNs are filtered out. AEB on Title resulted in **636** tuples; This low number is the result of different representations of titles as mentioned above. AEB on Author Name resulted in **8448** tuples; which is a good number and much more than 1500 and This is due to the fact that same author might have written multiple books. Because of these stricter blockers, we union the results from these equivalence blockers. The union of results of these three blockers results in **8486** tuples.

2.2 Overlap Blocker on Original_Title

With the help of **debug blocker** of **Magellan's Entity Matcher**, we analyzed the candidate tuples that we were missing by using AEB. Many of the titles that were of same book but had variation (as mentioned in previous section), were being removed by AEB. Thus, to include such candidate tuples, we added an Overlap Blocker at *word level* and $overlapsize = 5$ on the source tables. Note that we are stipulating that Title's should overlap with 5 words. Let's call the resulted candidate set after blocking be C_4. The results (union) of AEB and C_4 is used to perform further blocking in this step; Let's call this intermediate result C_{ob} with $\{C_{ob} = C_1 \cup C_2 \cup C_3 \cup C_4\}$ with C_1, C_2 and C_3 being candidates from AEB on ISBN, Title and Author respectively. The number of candidate tuples resulted from the union is **8792**.

2.3 Rule Based Blocker

The next two stages of blocking is done on the candidate tuples C_{ob} (not on source directory). To reduce the candidate tuples, we ran Rule Based Blocker on Author attribute such that any tuple with $Author\_lev\_sim < 0.8$ is blocked. Threshold (0.8) is kept high so that only names which differ with slight variations are only allowed. Number of candidate tuples, $|C_{ob}|$, that survived this step is **8497** and let's call it C_r.

2.4 Overlap Blocker

To further bring down the candidate set C_r, we run a simple Overlap Blocker on Title of the book again at *word level* and $overlapsize = 1$ and we excluding stop words from being considered for overlap. This makes sure that, in a candidate tuple's Book names, *there is at least one word that is not a stop word is*

matching between them. This stages reduces number of candidate tuples, $|C_r|$, from 8497 to **2327**. This is the final candidate set and let's call this final set of C.

2.5 Golden Data

This is the sample dataset that we labeled and used in the following steps. We picked a sample portion, using `em.sample_table()`, from the candidate tuple pairs C that survived after blocking. Then, manually labeled them with a value '1' if it is a match and '0' if it isn't.
Number of tuple pairs in Golden Data - **501**

3 Matching

We directly used the features produced by Magellan to run the first iteration on the entity tuple pairs. Following are the values of precision, recall, F1 that we obtained -

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x10cdcf3d0>	5	0.588235	0.700000	0.679245	0.721311	0.721311	0.682021
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x10cdcf410>	5	0.760000	0.640000	0.688525	0.653846	0.689655	0.686405
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x10cdcf490>	5	0.754098	0.787879	0.686567	0.666667	0.745763	0.728195
3	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x10cdcf250>	5	0.631579	0.551724	0.733333	0.666667	0.642857	0.645232
4	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x10cdcf390>	5	0.620690	0.555556	0.761905	0.627451	0.655172	0.644155
5	NaiveBayes	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x10cdcf1d0>	5	0.333333	0.400000	0.564103	0.647059	0.553191	0.499537

Figure 1: F1 scores after first run

As can be seen, we were let down by Magellan (unlike the explorer). We started debugging it. First thing we noticed is that, no features were being generated for 'Original_Title', which is the most important attribute for matching our candidate tuples. *(Imagine if Magellan went around the world without a mariner's compass)*

Then we had to manually add features for the attribute 'Original_Title'. First, we decided to perform Jaccard similarity function on 'Original_Title'. After adding this feature, the accuracy values increased a little. Since we didn't have any features on the most important measure before, we didn't pick any specific matcher. So we ran the test on all the matchers again. We obtained the following results -

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x10cdcf3d0>	5	0.754717	0.857143	0.888889	0.862745	0.869565	0.846612
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x10cdcf410>	5	0.836364	0.736842	0.851852	0.938776	0.793651	0.831497
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x10cdcf490>	5	0.754098	0.776119	0.686567	0.666667	0.745763	0.725843
3	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x10cdcf250>	5	0.840000	0.847458	0.888889	0.851064	0.771930	0.839868
4	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x10cdcf390>	5	0.836364	0.819672	0.945455	0.893617	0.825397	0.864101
5	NaiveBayes	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x10cdcf1d0>	5	0.333333	0.577778	0.714286	0.685714	0.793103	0.620843

Figure 2: F1 scores after adding Jaccard sim function for Original Title

We got better values this time as we considered 'Original_Title'(*duh!*). Since there was only one

feature on **'Original_Title'**, we decided to add a couple more to make the comparison stronger. Following are the new similarity functions that we used to add features on 'Original_Title' -

- Overlap Coefficient
- Smith Waterman
- Jaro Winkler
- Exact Match
- Needleman Wunsch

After adding the above features, we had an immense increase in the values of precision, accuracy and F1. We got the following results -

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean score
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x10ccc7410>	5	0.872727	0.888889	0.909091	0.884615	0.927536	0.896572
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x10ccc7390>	5	0.909091	0.903226	0.909091	0.938776	0.911765	0.914390
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x10ccc7450>	5	0.750000	0.794521	0.732394	0.730159	0.815789	0.764573
3	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x10ccc7190>	5	0.909091	0.923077	0.947368	0.978723	0.944444	0.940541
4	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x10ccc7310>	5	0.909091	0.906250	0.962963	0.916667	0.925373	0.924069
5	NaiveBayes	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x10ccc7110>	5	0.695652	0.754717	0.857143	0.850000	0.793103	0.790123

Figure 3: F1 scores after adding more similarity functions on Original Title

As evident from the values in the above table, we reached more than **90%** score for at least 3 matchers. We chose matcher based on **Linear Regression**, as it gave better scores than the other matchers. Since we had already reached desired scores, we proceeded with next steps.

Results of final run on Set J -

Matcher	Linear Regression
Precision	**93.46%**
Recall	**95.97%**
F1 Score	**94.7%**

Table 3: **Final Scores for Extraction on Testing Dataset J**

3.1 Recall

For our experiment, Recall value already exceeded expectations. There wasn't a necessity to increase it further.

4 Time Estimates

Approximate time estimates

- Blocking - Courtesy of the incomprehensible errors of Magellan, we had to spend a lot of time in this stage. We spent at least 5 hours to overcome the errors and perform actual blocking.

- Labeling the Data - It took us around 30 mins (x 2 people) i.e. 1 man hour to label the data and verify it.
- Finding the Best Matcher - It took us nearly 2 hours to add the features and find the best matcher.

5 Feedback on Magellan

Magellan, without a doubt, was a great help especially for debug blocking; At every step, debug blocking helped us understand what are the tuples that have high similarities (many of them even same books) and were being missed out by our blocking. Magellan has a lot of features to be praised for, but we limit ourselves to give feedback on what issues we found and what can be improved.

5.1 Error Messages

The error messages are mostly generic for certain kinds of issues. We had to go into complete Sherlock mode to understand Magellan. Eg: "**Assertion Error: Candset does not satisfy foreign key constraint with the right table**" was the error displayed when the type of the foreign key attributes were not matching with the base table. The error was vague and hard to trace back. The issue was that, one of the source's ISBN was being treated as *string* and the other source's ISBN as *integer*. We were not able to even force the type as well (explained in next section). Because of this, we had to make workaround of inserting one test tuple with ISBN being a string (not number like regular ISBN). We nearly spent around 4 hours to debug the issue. If the error had be more descriptive like "type mismatch" or "Value not found in the parent table", it would've been much helpful. We faced this issue again when we had missed adding the test tuple in both the source data and the same error appeared though the types were same. We had to manually go through the source data files and stare at it.

5.2 Setting the attribute type

As described in the previous section, we had issues with tables having ISBNs of different types. Goodreads source had ISBN's type as *string* and Barnes&Noble's as *integer*. Once we realized the issue, we tried to force the type by converting integer to string. We did not come across any function to set the attributes of the table to the desired data type although Magellan does offer function that lists the attribute type (`em.get_attr_types()`). If we had this options to set the attribute type (for possible attributes), then we wouldn't have faced the issue in the previous section

5.3 Feature set is not generated for all attributes

When we tried to automatically generate feature set for blocking (`get_features_for_blocking()`) as well as matching (`get_features_for_matching()`) both of these did not generate feature set for our main attribute 'Original_Title'; This is the deciding attribute on which we needed the feature set. Though Magellan was a great help every other time, we had to manually to add features for our project which is quite time consuming.

5.4 Black Box Blocker

As the features were not being generated automatically for blocking on 'Original_Title', we resorted to writing a Black Box Blocker ourselves. We wrote blocker which checks for prefix matching (e.g.,"Harry Potter and the Deathly Hallows" is a prefix of "Harry Potter and the Deathly Hallows (Harry Potter Series #7)") and if that doesn't work, we use a Dictionary to count the number of words matching to decide weather to drop a tuple or not. Adding to our woes, this blocker took **ages** to run; It became hard and time consuming to debug. We had to drop the idea of Black Box due to this reason. We do not understand the reason behind this.