Unification of Source Tables

Sowrabha Horatti Gopal sgopal3@wisc.edu
Pradeep Kashyap Ramaswamy pradeep.kashyap@wisc.edu
Sharath Prabhudeva Hiremath shiremath@wisc.edu

1 Source Tables

Our entity is a tuple containing **Book Details**. We have collected 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.
The following table gives the schema of the data collected 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

Since we had to keep the schema same for both tables to perform blocking and matching in Magellan, we had settled upon the a common schema for both the data sources.
Common Schema of both the data sources to perform blocking and matching (Table 2) -

Attributes
Original_Title
Author
ISBN13
Publication
Published_Date

Table 2: Schema used for Magellan

We did not add any other new table as our Goodreads data had enough information. We did not want to throw away the remaining attributes from the Goodreads data[1] that our **spider** had scraped with love. We believe that some of the remaining attributes (Ratings, Genres, Language, Reviews, Language) will be of value for us while extracting insights from our data.

[1] *We love our data more than Zuckerberg loves yours* ;)

2 Combination

Magellan helped us in the process of combination. Using Magellan, we had zeroed in on Linear Regression Matcher with custom features developed in the previous stage of the project. Then we applied this matcher to the candidate tuple pairs that we had obtained after the blocking step, to get the matching pairs. Finally we obtained the keys (which is ISBN13 in our case) from both the data sources for the matching pairs.

2.1 Final Table Schema

As mentioned in Section 1, we decided to enrich our data with the extra information that we had in the Goodreads data. So we settled on the schema of the final table which is as follows -

Attributes
Title
Author
ISBN13
Publication
Published_Date
Ratings
Genres
Reviews
Average Rating
Language

Table 3: Final Schema for Table E

2.2 Issues

As we had cleaned our data in the previous stage itself, there weren't any severe issues. There were some minor things that irked us though -

- Since Magellan gave us only the keys & predictions and not all the attributes associated with the tuple pairs, we had to read all the data again. This might be OK for smaller datasets, but we believe it will be nice of Magellan if it could provide us attributes as well.
- In our specific case, genres field was a list in Goodreads data. We had to decide the best way to add it in the final table. It wouldn't really help if we just add it as a list in the table, so we just parsed it to a string separated by commas.

2.3 Processing

To get the final table, we read the source data for Goodreads from the json we had extracted, as it included many more attributes useful for insights. For Barnes & Noble, we just used the csv that we used in the entity matching step, as there was nothing more to it. Following are the processing rules that we used to merge the two source tables listed Attribute-wise

- **Title :** To pick the title among the two tables, we used rule based on the length of title. We pick the title with the longest length. For majority of book titles, the longer title means more descriptive title - inclusive of the name of the series, exclusive of abbreviations.
  For example - One source table has the title of the book as "Fool's Errand", but the other has it as "Fool's Errand (Tawny Man Series #1)". Clearly, the latter is more descriptive, so we pick it. (This example can be seen in the sample given in this doc in Table 4)
- **Author :** In case of author also, we pick the longest of the author names coming from the source tables. The same rule used for title holds good here as well.

- **ISBN13 :** Since ISBN13 is uniquely associated with a particular book, intuitively we thought of picking it from table from which we picked the title.
- **Publication :** There may be different editions published for the same book, but by different publishers. But each edition of the book is again uniquely associated with ISBN13. So we pick the name of the Publication House from the source table from which we picked the ISBN13 (which is same as the one from where we picked Title).
- **Published Date :** We always pick the published date from Barnes & Noble source table. In the Goodreads data, published date is a list and of variable length, so we choose this attribute from Barnes & Noble table where it is already formatted.

All the following attributes are only present in Goodreads data, so they are always picked from **Goodreads**. (*Life is so much simpler when there are no other options*)

- **Ratings :** Goodreads
- **Genres :** Goodreads
- **Reviews :** Goodreads
- **Average Rating :** Goodreads
- **Language :** Goodreads

Since we had gotten rid of all the data with null values (which was very minimal), we didn't have to worry about 'void of nothingness' for any fields. *Moral :* Pre-processing is important!

Attribute	**Processing Rule**
Title	The one with the longest length
Author	The one with the longest length
ISBN13	Take from the source you picked title from
Publication	Take from the source you picked ISBN from
Published Date	Always pick from B&N, because it is stored as list of variable length in Goodreads json
Ratings	Always pick from Goodreads
Genres	Always pick from Goodreads
Reviews	Always pick from Goodreads
Average Rating	Always pick from Goodreads
Language	Always pick from Goodreads

Table 4: Summary of Processing Rules

3 Statistics

Number of tuples present in final combined table E - **1268**

3.1 Sample

Table 5 gives 6 sample rows from the final combined table.

ID	Title	Author	ISBN13	Publication	Date	Ratings	Genres	Reviews	Avg_Rtg	Lang
1	Fool's Errand (Tawny Man Series #1)	Robin Hobb	9780553582444	Random House Publishing Group	2002	"52,825"	"Fantasy, Fiction, Fantasy, High Fantasy, Science Fiction Fantasy"	"1,157"	4.28	English
2	The Yiddish Policemen's Union	Michael Chabon	9780007149827	HarperCollins	2008	"50,351"	"Fiction, Mystery, Science Fiction, Science Fiction, Alternate History, Mystery, Crime"	"5,860"	3.69	English
3	Unless	Carol Shields	9780007154616	Fourth Estate (GB)	2006	"10,719"	"Fiction, Cultural, Canada"	877	3.63	English
4	Remarkable Creatures	Tracy Chevalier	9780007178377	HarperCollins Publishers Ltd	2010	"31,582"	"Historical Fiction, Fiction, Historical, European Literature, British Literature, Literature, 19th Century, Adult Fiction, Womens, Audiobook, Book Club, Adult"	"3,674"	3.8	English
5	The Winter Rose	Jennifer Donnelly	9780007191321	Harper Collins	2009	"14,609"	"Historical Fiction, Romance, Historical, Fiction, Romance, Historical Romance"	"1,197"	4.3	English
6	Genghis: Bones of the Hills (Khan Dynasty Series #3)	Conn Iggulden	9780385342803	Random House Publishing Group	2010	"12,327"	"Historical Fiction, Fiction, Historical, War"	363	4.32	English

Table 5: Sample data from Final Table E

4 Soul of Stage 4

Here's is the python script that we used to merge the tables -

```
import json
import csv, sys

f1 = open('Matches.csv')
matches = csv.reader(f1)

# Create a dictionary for goodreads data with ISBN13 as the key
# This dictionary will be used to access information about
# matched pairs on goodreads side
gr_dict = {}

with open('../Phase3/test/goodreads.json') as file:
        data = json.load(file)

for item in data:
        #print item
        try:
                gr_dict[item['ISBN13'].encode(encoding="utf-8")] = item
        except KeyError:
                        continue

'''#Test the json is being read
gr_count = 0
for key in gr_dict:
        gr_count += 1
        #print str(gr_count) + " : " + key + " : " +
    str(gr_dict[key]['Original_Title'].encode(encoding="utf-8"))
print gr_count
'''

# Create a dictionary for barnes and noble data with ISBN-13 as the key
# This dictionary will be used to access information
# about matched pairs on barnes and noble side
ban_dict = {}

f2 = open('../Phase3/data/barnes_and_noble.csv')
ban = csv.reader(f2)

''' Barnes and Noble CSV schema
index           Column
0                       Original_Title
1                       Author
2                       ISBN13
3                       Publisher
4                       Published_Date
'''

for row in ban:
        try:
                ban_dict[row[2]] = row
        except KeyError:
```

```
                        continue
'''
#Test the barnes and noble data is being read and stored in dict
ban_count = 0
for key in ban_dict:
        try:
                ban_count += 1
                print str(ban_count) + " : " + key + " : " + str(ban_dict[key])
        except KeyError:
                        continue
print ban_count
'''

with open('merged_table.csv', "wb") as file,
open('all_matched.csv', "wb") as file_all:
        csv_file = csv.writer(file)
        csv_file_all = csv.writer(file_all)

        matched_count = 0
        for row in matches:
                try:
                        matched_count += 1
                        gr_item = gr_dict[row[1]]
                        ban_item = ban_dict[row[2]]

                        key = matched_count
                        title = ""
                        title_from_gr = None
                        author = ""
                        isbn13 = ""
                        publication = ""

                        #Rule 1 for title : Pick the one which is longest
                        if len(str(gr_item['Original_Title']
                        .encode(encoding="utf-8"))) >= len(str(ban_item[0])):
                                title = str(gr_item['Original_Title']
                                .encode(encoding="utf-8"))
                                title_from_gr = True
                        else :
                                title = str(ban_item[0])
                                title_from_gr = False

                        #Rule 2 for author : Pick the one which is longest
                        if len(str(gr_item['Author']
                        .encode(encoding="utf-8"))) >= len(str(ban_item[1])):
                                author = str(gr_item['Author']
                                .encode(encoding="utf-8"))
                        else :
                                author = str(ban_item[1])

                        #Rule 3 for ISBN-13: Take ISBN-13 from the matched keys
                        #obtained from Magellan from
                        #the source you picked title from
```

```
    #Rule 4 for Publisher: Take from the source you
    #picked title from
    if title_from_gr :
        isbn13 = row[1]
        publication = str(gr_item['Publication']
        .encode(encoding=”utf−8”))
    else :
        isbn13 = row[2]
        publication = str(ban_item[3])

    #Rule 5 for Published Date: Always take from BAN
    published_date = str(ban_item[4])

    #Rule 6 for Ratings, Genres, Reviews,
    #Average Rating, Language: Always pick from Goodreads
    #as none of them are present in BAN
    ratings = str(gr_item['Ratings'].encode(encoding=”utf−8”))
    reviews = str(gr_item['Reviews'].encode(encoding=”utf−8”))
    avg_rating = str(gr_item['Average_Rating']
        .encode(encoding=”utf−8”))
    lang = str(gr_item['Edition_Language']
        .encode(encoding=”utf−8”))
    genres = gr_item['Genres']
    genre = ””
    i = 0
    for g in genres:
        if i == (len(genres) − 1):
            genre += str(g.encode(encoding=”utf−8”))

        else :
            genre += str(g.encode(encoding=”utf−8”))
            genre += ”,␣”
        i += 1

    # Write to the file which contains all matched tuples
    #from source tables
    csv_file_all.writerow([matched_count, row[1],
        str(gr_item['Original_Title']
        .encode(encoding=”utf−8”)),
        str(gr_item['Author'].encode(encoding=”utf−8”)),
        str(gr_item['Publication']
        .encode(encoding=”utf−8”)),
        str(gr_item['Published_Date']), row[2],
        str(ban_item[0]), str(ban_item[1]),
        str(ban_item[3]), str(ban_item[4])])

    # Write to the final table E
    csv_file.writerow([matched_count, title, author, isbn13,
        publication, published_date, ratings, genre,
        reviews, avg_rating, lang])
except KeyError:
    matched_count −= 1
    continue
```