CS 784

Advanced Topics in Database Management Systems

Report on

Class Project

Team

TRIBUTARY

where data seamlessly integrates…

Rohith Subramanyam
rohithvsm@cs.wisc.edu
Heemanshu Suri
hsuri@cs.wisc.edu
Department of Computer Sciences
University of Wisconson-Madison
Fall 2014

Index

1) Sources

The 2 sources chosen were books from Amazon and Barnes & Noble. Since, the collection of books in these websites is very extensive, we restricted our crawler to the following 3 categories of books as we were looking for around 3000 tuples each:

- Fiction and Literature
- Engineering
- Computer and Technology

1.1) Amazon

is a leading e-commerce website to do shop online.

1.1.1) Data acquisition

Wrote a python program to parse the HTML pages from www.amazon.com. Please find a very brief man documentation of the program below:

```
$ ./tableA_amazon.py --help
usage: tableA_amazon.py [-h] [--version] [-n [NUMPRODUCTS]] urls [urls ...]

This module parses the Amazon product search pages and parses each product
page in the results to extract details about the product. In this case the
product is books.

positional arguments:
  urls                  The URLs of the product search pages from which the
                        products are parsed

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -n [NUMPRODUCTS], --numproducts [NUMPRODUCTS]
                        The number of products to parse. [default: 5000]
```

As you can see from the help message above, the program accepts one or more amazon product search results urls as positional arguments and an optional argument of the total number of products attributes to collect. Find a brief pseudocode of the program below:

```
for product search results url in the product search results urls (command-line argument):
    while you can paginate in the product search url:
        navigate to each product page and extract the product attributes
            if the total of products has reached numproducts, stop!
```

The program mainly makes use of the following python libraries:

- **urllib2:** a standard python module which helps in opening URLs (mostly HTTP)
- **bs4 (Beautiful Soup 4):** is a third-party library designed to perform screen-scraping

1.1.2) Number of pages parsed

Within a product search results page, the products are sorted by “New and Popular” type. In amazon, any product search returns results in 100 pages with 12 results in each page. Hence, the total number of parsed HTML pages is shown below:
Product search pages: 100 * 3 = 300
Product pages = 300 * 12 = 3600
Total = 3600 + 300 = 3900 pages

1.1.3) Sample pages

Sample product search results page: http://www.amazon.com/b/?node=17

Sample product page: http://www.amazon.com/What-If-Scientific-Hypothetical-Questions-ebook/dp/B00IYUYF4A/ref=lp_17_1_1?s=books&ie=UTF8&qid=1418949914&sr=1-1

1.2) Barnes & Noble

is the largest retail bookseller in the United States, and the leading retailer of content, digital media and educational products in the country (taken from Wikipedia).

1.2.1) Data acquistion

Wrote a python program to parse the HTML pages from www.barnesandnoble.com. It makes use of the same libraries as mentioned above for Amazon.

1.2.2) Number of pages parsed

Around 3490 HTML pages were parsed from www.barnesandnoble.com

1.3) Source URLs

source	Fiction and Literature	Engineering	Computers and Technology
amzn	http://www.amazon.com/b/?node=17	http://www.amazon.com/b/?node=173515	http://www.amazon.com/s/ref=lp_283155_nr_n_7?rh=n%3A283155%2Cn%3A%211000%2Cn%3A5&bbn=1000&ie=UTF8&qid=1414297367&rnid=1000
bnn	http://www.barnesandnoble.com/s/?aud=tra&dref=9&fmt=physical&sort=SA&startat=1&store=BOOK&view=grid	http://www.barnesandnoble.com/s/?aud=tra&dref=13&fmt=physical&sort=SA&startat=1&store=BOOK&view=grid	http://www.barnesandnoble.com/s/?aud=tra&dref=25&fmt=physical&sort=SA&startat=1&store=BOOK&view=grid

2) Data Extraction

2.1) Schema

Both the tables have the same following schema:

attribute name	attribute type
id	INTEGER
ISBN_13	TEXT
Title	TEXT
Author	TEXT
Price	FLOAT
Publisher	TEXT
Publication_date	TEXT
Pages	INTEGER

2.2) Sample tuples

2.2.1) amazon


```
{
  "id": 1,
  "ISBN_13": “9780307588371”,
  "Title": "Gone Girl: A Novel",
  "Author": "Gillian Flynn",
  "Price": 8.97,
  "Publisher": "Broadway Books",
```


```
    "Publication_date": "04/22/2014",
    "Pages": 432
}
```


2.2.2) Barnes & Noble


```
{
    "id": 3,
    "ISBN_13": "9780321968135",
    "Title": "Exposure: From Snapshots to Great Shots",
    "Author": "Jeff Revell",
    "Price": 17.51,
    "Publisher": "Peachpit Press",
    "Publication_date": "03/24/2014",
    "Pages": 264
}
```


2.3) Data Extraction

The attributes for the book were extracted looking for certain tags in the HTML page of the book. For instance, below is an explanation of how it was done for amazon:

- Title: tag with id="productTitle" or id="btAsinTitle"
- Price: tag with class="a-color-price"
- Author: <a> tag with class="contributorNameID
- Publisher and Publication_date: <b> tag with id="detail-bullets" and text "Publisher"
- ISBN_13: <b> tag with text "ISBN-13"
- Pages: <b> tag with id="detail-bullets" and text 'pages'

Similarly, for Barnes & Nobles, we looked for HTML tags defining the book attributes.

2.3.1) Challenges

Some challenges faced while extracting the attributes is listed below:

- Amazon has the same book available in different forms like print, kindle edition, digital copy, etc. Each has different prices. Some of them are available only in kindle edition or in print form. So to extract the price, we had to look for certain tags in the absence of another
- Extracting multiple authors
- Some books had ISBN_10 instead of the new standard ISBN_13. Solved it by having looking for ISBN_10 in the absence of ISBN_13

2.3.2) Data pre-processing

- Store the price as a float by removing '$' symbol
- Try to store the date in a consistent format: MM/DD/YYYY. But some books had only the month and year as the publication date, in which case, convert it to MM/YYYY
- Remove special characters like '-' in the ISBN_13 field and keep only digits

3) Blocking

Wrote a python program to perform blocking (blocking.py). It accepts 2 files containing data in json format and a matching attribute on which the blocking has to be performed.
Steps:
1) An inverted index, indexed on the blocking attribute is built from data in table B.
2) It then iterates through all tuples in the table A
pairs it with table B tuples having the same value for the blocking attribute

Because, table B is indexed on the blocking attribute, looking up tuples on it is efficient compared to a brute force method of trying to match every record in table A with each record in table B which would be in the order of n^2. Therefore, the matching records in table C are pairs of records from table A and table B having the same value for the blocking attribute.

All the blocking strategies tried are listed below:

1. **Blocking on Publisher**
   Publisher was used to perform blocking using exact match. Below is the statistics of the blocking operation:
   no. of records in table A: 3467
   no. of records in table B: 3253
   no. of records in the candidate set, table C: 81294
   The pair of records to be matched is thus reduced from (3467 * 3467 = 11278151) to 81294 which is 0.72 % of all the possible pairs.

2. **Blocking on Publisher and Publication year and month**
   While picking a random sample of around 300 tuples from the above tableC for the golden data, we noticed that the number of true positives picked was very less (< 20). We wanted to work with a bigger set of true positives. Hence, we decided to reduce the size of tableC.

   After blocking with Publisher, we included a pair of tuples in tableC only if they had the same publication year and month. Statistics of this blocking operation is shown below:
   no. of records in the candidate set, table C: 1099
   The pair of records to be matched is thus reduced from (3467 * 3467 = 11278151) to 81294 which is ~0.01 % of all the possible pairs.

4) Golden Data

Wrote a python program, sampler.py, which randomly samples ‘n’ number of tuples from a csv file. At this stage, we added a capability to the our blocking program, blocking.py in the presence of an additional command-line argument to label the tuple pairs if their ISBN_13 attributes match exactly. And this labeled tableC was fed to the sampler.py program above to pick a random sample of 300 tuples from it (tableG).
Wrote another program, sample_generator.py to repeatedly run the above sampler and count the number of tuple pairs with label 1 and stop when the count is beyond 40 because we wanted to work with at least 40 true positives in EMS.

4.1) Challenges

Some tuples pairs had the same ISBN value but were of different editions. They had a considerable dissimilarity in their title because of words like (Edition, One, I, II, etc.) and moreover had different publication dates as well. They were getting labeled as true or 1 because of the same ISBN even though they are not matches. We ignored such few cases as a data problem because in real-world scenarios, it is highly unlikely that all the data will be perfect.

5) The Matching Step

5.1) Trial 1

5.1.1) Feature

Created the following feature using Feature menu → Add Feature:
title_jaccard: title attribute of the 2 tables using JACCARD function

Name	Function	Argument1	Argument2
title_jaccard	JACCARD	amazon.Title	bnn.Title

5.1.2) Rule

Created the following rule using Rule menu → Add Rule:
title_jaccard_rule: title_jaccard >= 0.85

Name	Table 1	Table 2	Rule String
title_jaccard_rule	amazon	bnn	title_jaccard >= 0.85

5.1.3) Matcher

Created a matcher using the Matcher menu → Add Matcher:
title_jaccard_matcher: title_jaccard_rule

Name	Table 1	Table 2	Matcher String (Short)	Matcher String (Long)
title_jaccard_matcher	amazon	bnn	title_jaccard_rule	title_jaccard >= 0.85

5.1.4) Evaluate

Evaluate menu → Using Labeled Data:
Precision recall achieved:
Precision = 1
Recall = 0.55

5.1.5) Debug

Debug menu → Using existing results:

Matching Summary

Evaluation summary:

Precision:	1
Recall:	0.55
F1 score:	0.71
Number of precision errors:	0
Number of recall errors:	19

Matching summary:

All pairs in the candidate set:	300
Total matches:	23
Total rules:	1

Rule summary:

title_jaccard_rule matches	23

5.1.6) Observations

The 2 tables had slight variations in the title names resulting in a lower recall. For example:
Amazon title: The Light Between Oceans: A Novel
Barnes & Nobles title: The Light Between Oceans
Though refer to the same book (in fact have the same ISBN), due to the slight variations in the way the title is stored in each website the jaccard score on this example was 0.67
These cases were contributing to the lower recall value of 0.55.

5.2) Trial 2

5.2.1) Objective

In order to improve the recall due to the variations in title, we decided to experiment with the author name. Since a single author can have multiple books, we decided to have a conjunction of author and Publication_date.

5.2.2) Feature

Added the following features:

- **author_exact:** author attribute of the 2 tables using JACCARD function
- **date_exact:** Publication_date of the 2 tables using EXACT_MATCH function

date_exact	EXACT_MATCH	amazon.Publication_date	bnn.Publication_date
author_jaccard	JACCARD	amazon.Author	bnn.Author

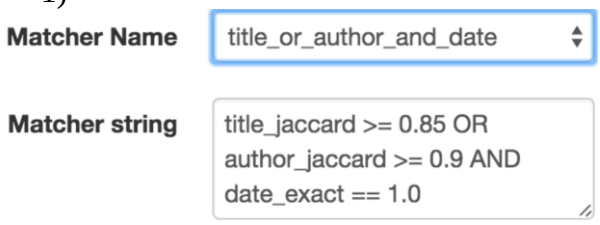
5.2.3) Rule

Added the rule as follows:
author_date: author jaccard >= 0.9 AND date exact == 1

author_date	amazon	bnn	author_jaccard >= 0.9 AND date_exact == 1.0

5.2.4) Matcher

Created a matcher as follows:
title_or_author_and_date: title jaccard >= 0.85 OR (author jaccard >= 0.9 AND date exact == 1)



5.2.5) Evaluate

Evaluate menu → Using Labeled Data:
Precision recall achieved:
Precision = 1
Recall = 0.64

5.2.6) Debug

Debug menu → Using existing results:

Matching Summary

Evaluation summary:

Precision:	1
Recall:	0.64
F1 score:	0.78
Number of precision errors:	0
Number of recall errors:	16

Matching summary:

All pairs in the candidate set:	301
Total matches:	29
Total rules:	2

Rule summary:

title_jaccard_rule matches	26
author_date matches	6

5.2.7) Observation

There was a slight improvement in the recall from 0.55 to 0.64 but still much left to be desired. On debugging, we found that there were certain true labeled records that were mismatched due to the Publication_date exact match function. On inspecting further we saw that it was due to inconsistencies in the way the date was formatted in Amazon and Barnes & Nobles. For example:
Amazon mostly had Publication_date in the following format 09/08/2013
Where as most of the records in Barnes & Nobles were using single digit representation for months January to September and dates from 1st to 9th. Example: 9/8/2013.

5.3) Trial 3

5.3.1) Objective

Because of the above observations with regards to the way dates were formatted, we decided to format the dates in a consistent manner in the table data.
We wrote a python program to format the dates in Barnes & Nobles to MM/DD/YYYY.
After this we added this data again as table 2 using Table menu → Import from .csv file

5.3.2) Evaluate

Evaluate menu → Using Labeled Data:
Then, we matched using the same matcher as used above, title_or_author_and_date.
Precision recall achieved:
Precision = 1
Recall = 0.82

5.3.3) Debug

Debug menu → Using existing results:

Matching Summary

Evaluation summary:

Precision:	1
Recall:	0.82
F1 score:	0.9
Number of precision errors:	0
Number of recall errors:	8

Matching summary:

All pairs in the candidate set:	301
Total matches:	37
Total rules:	2

Rule summary:

title_jaccard_rule matches	26
author_date matches	23

5.3.4) Conclusion

With this, we achieved a satisfactory recall value of 0.82.
The false negatives were mainly because, we labeled the golden data based on the ISBN. And a few pairs had the same ISBN but were actually different editions. And, hence did not either match the title rule and Publication_date. And, in a few cases, Amazon and Barnes & Nobles had slight variations in their Publication_date in terms of a few days. For example, 10/13/2012 in Amazon and 10/17/2012 in Barnes & Nobles.

6) Feedback

Experience using EMS

- Nifty tool to perform the hard task of data matching in an intuitive way
- Analyst is in total control of the matching process: really nice way in which a human can tweak whenever and wherever required and the system does the rest of heavy lifting
- The rules, features and matchers are all saved on disk and hence it is easy to reuse them while trying different matchers and while reloading the project at a later time
- Debugging using existing results is a very handy feature to get a peek into the records and see what is giving the resulting precision and recall value
- Debug tab of the application is very intuitive due to the color coding and the evaluated values of the rules displayed

Suggestions

- Capability to import data from other formats like json, tsv, xls, xlsx, sql loader files, etc.
- Delete functionality not working for all the entities: Project, Table, Feature, Rule, Matcher

- JACCARD did not appear in the Function Name dropdown while adding a feature for our Publication_date attribute which is a TEXT field
- Features can be defined using functions on non-text attributes like Integers, Float, etc. Functions like ==, !=, can be provided for such data types. In our case of books, pages attribute is an integer attribute and we would have liked to have a feature using the function "equal to" on it
- Add support for DATE data types and allow for comparison of dates as a function which can handle dates in different format
- Allow other logical operators like OR, NOT while adding multiple features in a rule. Also allow AND while creating a matcher with multiple rules. This can be somewhat achieved today with a combination Rules and Matchers. But, supporting all operators in both Rules and Matcher allows more flexibility to the analyst
- Ability to run some pre-processing function over attributes values before matching (the user need not cleanup the data before importing it into EMS)
- Support to match more than 2 tables
- Support for regex matching of attributes
- Editing features, rules and matchers UI is not similar to the UI when adding these. One has to manually edit the text defining them
- Start the EMS application without having to run some commands on the shell and then using the browser, like starting a desktop application.
- Run EMS on the cloud
- Interface to report bugs to the developer which will report some diagnostic information like the user environment, current stack frame, etc. to the developer to help debug

7) Pointers

Project home page: http://pages.cs.wisc.edu/~rohithvsm/cs784/ contains a readme file and a directory for each stage.

7.1) Stage2:

Raw data:

- **tableA_amazon.json**
- **tableB_b_and_n.json**

Crawlers:

- **tableA_amazon_py.txt**
- **tableB_b_and_n_py.txt**

HTML pages:

- **tableA_amazon_html**
- **tableB_b_and_n_html**

7.2) Stage 3

blocking code: **blocking_py.txt**
blocking explanation file: **blocking-explanation.txt**
tableC: **tableC.csv**.

7.3) Stage4

Golden data: **tableG.csv**
Matcher used: **stage4.pdf**.
The final Precision and recall values achieved is 1 and 0.82 respectively.