Meta open linked books:

a no-hype recommender system.

Project Defense

Víctor Herrero, Diego Garcia-Olano, Elsa Mullor

Table of Contents

Book Recommender

- We wanted to make a system

  which could leverage the linked nature/richness of the data at **dbpedia.org**

  while also leveraging the social nature/user ratings of **goodreads.com**
- We handle **cold start** situations, when we don't know who a user is, or a user has no history (ie, has left no reviews/ratings)
- We provide **user based collaborative** recommendations
- We also provide **content based** recommendation

Data Sources

Data Sources

- **GoodReads**
- Provides with an API:
  - Provides lots of information
  - The result can be either **XML** or **JSON** (depending on the API call)
- **DBpedia**
- Provides with an API:
  - Very poorly data retrieving
- Allows powerful query language for RDF:
  - **SPARQL**


```
- <GoodreadsResponse>
  - <Request>
      <authentication>true</authentication>
      <key>g1kKTmrhRCN8IpXI8elXQ</key>
      <method>search_index</method>
    </Request>
  - <search>
      <query>the catcher in the rye</query>
      <results-start>1</results-start>
      <results-end>20</results-end>
      <total-results>110</total-results>
      <source>Goodreads</source>
      <query-time-seconds>0.12</query-time-seconds>
    - <results>
      - <work>
          <books_count type="integer">291</books_count>
          <id type="integer">3036731</id>
          <original_publication_day type="integer" nil="true"/>
          <original_publication_month type="integer" nil="true"/>
          <original_publication_year type="integer">1951</original_publication_year>
          <ratings_count type="integer">1443681</ratings_count>
          <text_reviews_count type="integer">30162</text_reviews_count>
          <average_rating>3.77</average_rating>
        - <best_book type="Book">
            <id type="integer">5107</id>
            <title>The Catcher in the Rye</title>
          - <author>
              <id type="integer">819789</id>
              <name>J.D. Salinger</name>
            </author>
            <image_url>http://d.gr-assets.com/books/1398034300m/5107.jpg</image_url>
            <small_image_url>http://d.gr-assets.com/books/1398034300s/5107.jpg</small_image_url>
          </best_book>
```

Data Sources

- **Initial Data Sources**

- *DBbook_Items_DBpedia_mapping.tsv* — *retrieve books from external sources + store locally*

- *DBbook_train_binary.tsv* — *User initial Database +*
- *DBbook_train_ratings.tsv* — *Two ratings (binary/non-binary)*

- http://challenges.2014.eswc-conferences.org/index.php/RecSys#DATASET

Architecture

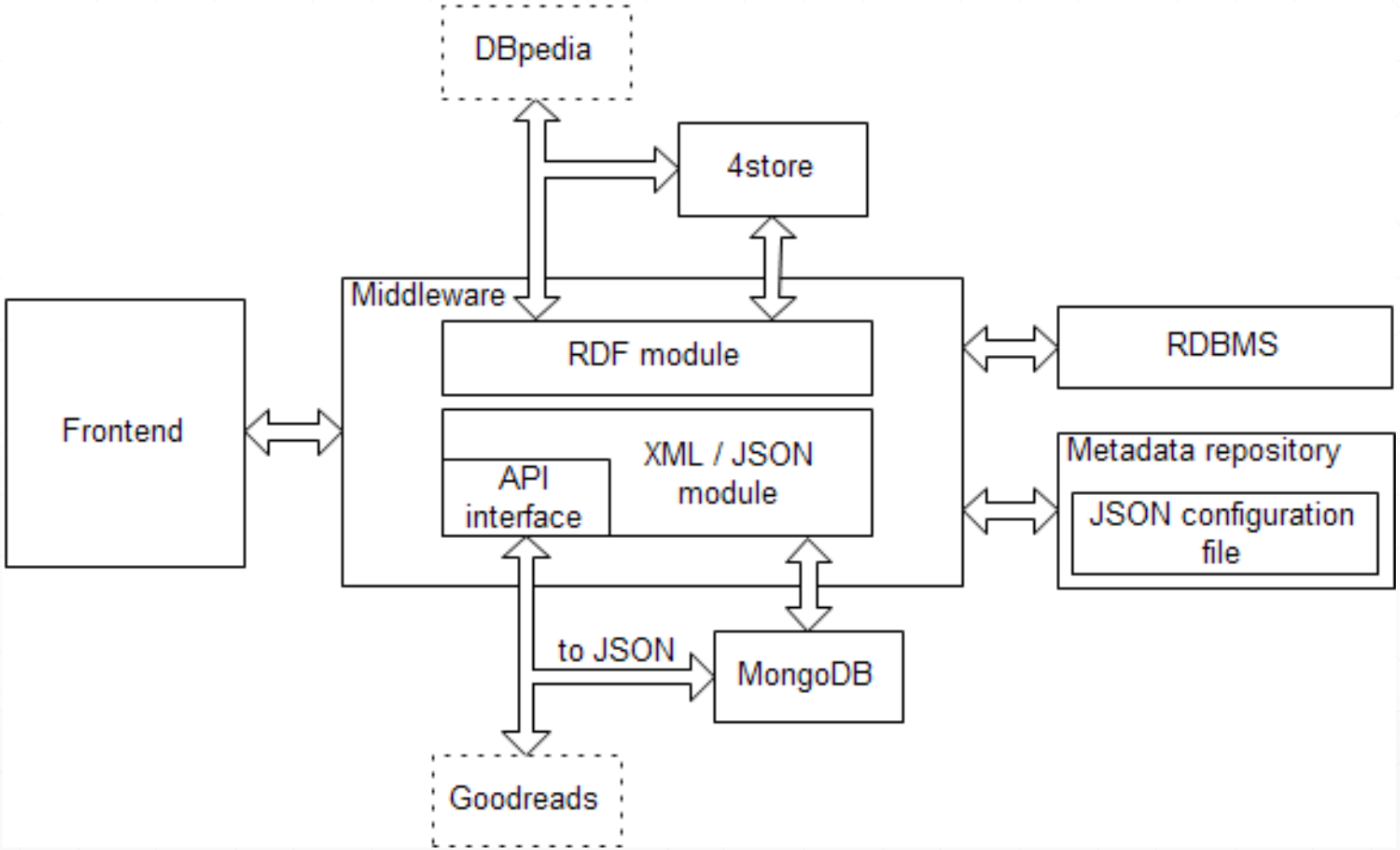
Architecture

- **GoodReads**
- JSON data stored in MongoDB
- MongoDB: acknowledged (huge community, lots of management tools)

Architecture

- **DBpedia**
- RDF data store in 4store
- Why 4store?
  - Light and native triplestore:
    - We did not want to run an RDFizer on top of a RDBMS
    - Other native solutions like Virtuoso were too **heavy** and **hardware-dependent**
  - Already provided with a **Python library**
  - Supports SPARQL queries

Architecture



Architecture

- Frontend : User Interface
- External Data Sources: Goodreads / DBpedia
- Local Data Sources : MongoDB / 4store
- Additionally an RDBMS
- Middleware
- Metadata Repository

Data Source	Querying technique
Goodreads	API
MongoDB	MongoDB interface
DBpedia, 4store	SPARQL
RDBMS	SQL

Integration Layer

Integration Layer

- The Middleware (Python)
- External Sources: Goodreads / DBpedia
- Local Sources: MongoDB / 4store
- An RDBMS
- Config file : Metadata Repository

Integration Layer

- **Goodreads** results:
- Evaluated, cleaned, selected
- Pick best results with evaluation of confidence and threshold it must pass.

  This option and threshold can be changed in metadata repository.
  1) Select based on **Confidence**
  2) If none found**,** Select based on **Direct hit of title**
     (and if there are multiple, select that with the most reviews)
  3) Select based on **Number of Ratings only**
     (normally bad, but occasionally great.)

Integration Layer

- **DBpedia** query:
  CONSTRUCT when retrieving full triples
  SELECT when retrieving concrete pieces of data

  } No syntax transformations are needed

- Select results:
  1. Book title **split** into words
  2. Run global query combining those words (**regular expression**)
  3. Assign **scores** based on similarities
  4. Finally, **sort** scores

Integration Layer

- **RDBMS**:
  - Book table (ID, title)
  - User table (userid, name)
  - Ratings tables (UserID, bookID, rating):

    one for Decimal ratings from 1 to 5 and one for Binary ratings
- Integration between RDBMS + MongoDB + 4store is achieved by means of IDs:
  - After agreeing on the matching of a book obtained externally (i.e., goodreads and dbpedia), why would I need to run the same matching process if that book is again requested?
    - So, an ID is just assigned to that book and then stored locally
    - The RDBMS can be seen as a cache of already integrated books (a corpus for ratings)
- Lookup in those tables to determine search case (cold case or not)

Integration Layer

- Integration of results: Comparing Title + Author

  Assumption: if two books have same title and author, they are the same

- If match from both sources agree: assume correct book found and return
- Otherwise return list of results from DBpedia

  Assumption: DBpedia is richer/linked so it has priority

- Results imported into local MongoDB after match with DBpedia and insertion into local RDBMS.
- These rules can be configured in the metadata repository (JSON file)

Metadata Repository

Metadata Repository

- Represented in a **JSON file**:
  - **Frontend**: What can the client search for?
  - **External Sources**:
    - List of external sources to add in the **Integration Layer**
      - **Searchby:** Map global to local concept used for search (e.g., title into URI)
      - **Matching:** Map global to local attribute used for matching between external sources.
      - **Mapping:** Map global to local concepts that must be returned to the Frontend
  - **Backend:**
    - Same keywords than external sources but refers to local DBMS
- **Search Rules**:
  - Additional rules (specify behavior in certain situations)
  - Describes the criteria used to define similarities between books (e.g., same author and genre)

```
"frontend":
{
    "searchby":["title","author"],
    "results":{"page":10},
    "metadata_to_display":["title","author","genre","year_of_publication",
                           "cover_image","rating","country"]
},
                    "external_sources":
                    {
                        "dbpedia":
                        {
                            "searchby":
                            {
                                "title":
                                {
                                    "title": { "value":"http://dbpedia.org/property/name", "main":true },
                                    "author": {  "value":["http://dbpedia.org/property/author",
                                                  "http://dbpedia.org/property/name"], "main":false  }
                                }
                            },
                            "mapping":
                            {   "title":"http://dbpedia.org/property/name",
                                "author":"http://rdf.recommender/authorName",
                                "country":"http://dbpedia.org/property/country"
                            },
                            "mapping_depth":1,
                            "index":"value",
                            "matching": { "title":"title", "author":"author" }
                        },
```

mappings

even more mappings

```
    "goodreads":
    {
        "searchby":
        {
            "title":"best_book.title",
            "author":"best_book.author.name"
        },
        "mapping":{ "cover_image":"best_book.image_url",
                    "year_of_publication":"original_publication_year.val",
                    "title":"best_book.title",
                    "author":"best_book.author.name"},
        "mapping_depth":0,
        "index":"",
        "confidence_threshold":0.5,
        "matching": { "title":"best_book.title", "author":"best_book.author.name" }
    }
},
"backend":
{
    "searchby":{"title":"booktitle", "author":"authorname"},
    "mapping":{"rating":"ratings.internal"}
},
"searchrules" :
{
    "dbpedia_as_master_external_source":true,
    "check_for_local_version_first":true,
    "goodreads_select_most_rated_as_best":false,
    "similarity":
    {
        "current":"author.genre",
        "criteria":
        {
            "author": [ { "name":"author", "value":"http://dbpedia.org/property/author" } ],
```

Metadata Repository

- Rules
- How to search (external sources, backend) : terms to query
- How to search (Special behavior: cold case/ perfect hit…)

Recommendations

Recommendations

- **Collaborative**: Distances on User
  (user-based, except cold start)
- **Assumptions**: 1 - Less users than books
  2 - Stable (updates once a week)
- High computational cost (number of books/users/ratings)

  In practice :pre-computed in the background and stored for quick lookup.

- **Content Based**
- Binary/non-binary ratings (different distances)

Recommendations

- Situations:
- **Cold Case**
  no user information,
  no query for a book :

  1) use local database,

  2) rank by ratings
  (weighted by # of reviews)

```
scores = []
max = 0
for item in tprefs:
    sc = 0
    numb = 0
    for persone in tprefs[item]:
        sc = sc + tprefs[item][persone]
        numb = numb + 1
    if max<numb : max=numb
    scores.append((float(sc)/numb,numb,item))

# Sort the list so the highest scores appear at the top
scores = [(ra*(5+nb/max)/6 ,nb,item) for ra, nb, item in scores]
scores.sort( )
scores.reverse()
return scores
```

Recommendations

- **User-history** : (User Information)
  Similar users (Find book user might like, that he has not read.)

Users	Sim	Book1	Sim * Book1	Book2	Sim*Book2	Book3	Sim*Book3
User1	0,99	3	2,97	5	4,95	2	1,98
User2	0,38	3	1,14			5	1,9
User3	0,56	4,5	2,52	2	1,12	3	1,68
Total			6,63		6,07		5,56
SimSum	1,93		1,93		1,55		1,93
T/SimSum			3,44		3,92		2,88

- Sim*Book: weighted rating (heavier if very similar user)
- Total/SimSum: Total score normalized by number ratings

Recommendations

- **Book-based** (No user information, A book query)
- Content-based: (author, title, genre…)
  - Using Linked RDF source (4store locally)
- User-based: “People who like this book also like…”
  - Transpose point of view
  - Compute Similarities
- Any combination (if have required information)

Live Demo !!

Thank you for your attention…

Bibliography

- [1] SEGARAN, Toby. *Programming collective intelligence: building smart web 2.0 applications*. " O'Reilly Media, Inc.", 2007.

- Technologies for web app: python, mysql, mongodb, 4store
- Code will eventually be placed on: www.github.com/diegoolano along with build requirements/installation notes.