**1. Introduction**

The goal of this project is to build a movies & actors portal to give the users the data of both movies and actors. By combining of the information, the portal can give more interesting information than just simply querying the database. For instance, the portal can build a crew of a movie of specific genre based on this information.

The procedure of build the portal is not a trivial work. The stages are: 1) Fetching Data: code crawlers to fetch useful information from different website; 2) Entity Resolution: to determine which data from different source refer to the same movie / actor; 3) Data Fusion: merge different data of the same entity; 4) Data Portal: code a website the demonstrate the final result of the merged data.

**2. Specification**

**2.1 Data Source**

All of the data of movies and actors are crawled from<http://themoviedb.org/> (TMDB),<http://www.imdb.com/> (IMDB) and<https://www.wikipedia.org/> (WIKI). All the data we fetched were gotten by using crawlers without any API.

**2.2 Entities**

**2.3 Classes in target schema**

**2.4 Entities contained in at least two datasets**

**2.5 Joint dataset attributes**

**2.6 Attributes contained in at least two datasets**

**2.7 Data File Format**

The file formats we use to store the data from crawlers are *JSON* and *XML*.

**2.4 Database**

We use one of the popular No-SQL, *MongoDB*, as the database to store the data. Comparing to other RDBMS, fixed schemas could not be assigned so that we can have a very flexible database schema to store the data from our sources. Thus, we can more concentrate on the procedure of data Integration without dealing with a lot of relation and schema problems.

**2.5 Programming Language**

All of our codes are programmed using *Ruby*, which is a popular dynamic programming language.

**3. Fetching Data**

**3.1 Tools We Used**

We depend on two Ruby Gems (library of ruby), Nokogiri and Httparty to build our crawler. Nokogiri provides the ability to locate the content of a specific tag of HTML/CSS, and Httparty gives the ability to easily send HTTP request and help to maintain the cookies. Regular Expression is also highly used when coding the crawler to extract the target character string.

**3.2 Crawling Strategy**

We use different crawling strategy when crawling different website. As for themoviedb.org, it is easy to crawl all the data sequentially since the website is organized simply. As for wikipedia.org, we crawl the data \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. The most challenging part is crawling the data from imdb.com. Since no complete list of movies or actors are available in the website, we have to crawl the data using Breath First Search given some original seeds.

**3.3 Raw Data Statistics**

We crawled 20,000+ movies and 20,000+ actors from themoviedb.org, 10,000+ movies and 11,000+ actors from imdb.com, and 5,000+ movies and 7,000+ actors from wikipedia.org. Raw data were stored in *JSON* or *XML* format files for further processing.

**4. Entity Resolution**

**4.1 Schema Unifying**

Before performing entity resolution, we eliminate the unwanted attributes of every data entry from the three sources, and design a unified schema for all of the data entries. The schemas of movies and actors are as follow:

**1) Schema: MOVIE**

|  |  |
| --- | --- |
| ATTRIBUTE | DATA TYPE |
| title | String |
| year | Integer |
| rating | Float |
| directors | Array |
| casts | Hash |
| main\_casts | Array |
| total\_time | Integer |
| languages | Array |
| alias | Array |
| country | Array |
| genre | Array |
| writers | Array |
| filming\_locations | Array |
| keywords | Array |
| match\_id | Integer |
| db\_name | String |

**2) Schema: ACTOR**

|  |  |
| --- | --- |
| ATTRIBUTE | DATA TYPE |
| name | String |
| birthday | Date |
| gender | String |
| place\_of\_birth | String |
| nationality | String |
| known\_credits | Integer |
| adult\_actor | Boolean |
| years\_active | String |
| alias | Array |
| biography | String |
| known\_for | Array |
| match\_id | Integer |
| db\_name | String |

**4.2 Data Cleaning and Re-storing**

In this step, we unify the data using the schema above. Since the attribute name and data type of data from different website is various, and some of the values may be in some unwanted condition (e.g. two space between first name and last name or &amp; instead of &), they are modified in this step. Besides, we can filter out some entries which are definitely not belongs to movies or actor (e.g. TV Show or writers).

After doing data cleaning in the main memory, all the data are stored in the database preparing for further processing, instead of stored in normal files.

**4.3 Methods and Algorithms of Entity Resolution**

**1) Naive Pairwise Matching:** simply use for loop to compare the matching score of different entries in different database (source). When the score is higher than some threshold, set the match\_id of them to be the same. However, the running time of naive pairwise matching is very high (O(m!\*n2). luckily, m = 3 in our project), we must use some theorem (e.g. transitivity) and blocking (clustering) to reduce the number (n) of elements in a set, to speed up the pairwise matching process.

**i) Decision Tree:** To determine whether two entries refer to the same entity, we firstly go down a decision tree which just has 3 levels. As for actor, we compare name and birthday; as for movie, compare title and year. If it cannot be determined by the decision tree, we will compute the matching score between the two entries and see whether it is over some threshold.

**ii) Pairwise Matching Score:** matching score is computed based on several major attributes of the data entry with weights (e.g. for actor: name, birthday, place\_of\_birth, known\_for). Some algorithms are used to compute the similarity of names, movie’s title and array.

**a) Jaro-Winkler:** an algorithm which highly suits computing the similarity of different names with one single string (i.e. last name).

**b) Monge-Elkan:** good for handle the problem with a name with multisection (e.g. Zhang San). By using this algorithm, we can identify an actor with his/her name even if it is rearranged (e.g. Zhang San == San Zhang). Inside Monge-Elkan, we can use single string algorithm such as Jaro-Winkler or N-Grams to suit the best condition.

**c) Jaccard Coefficient:** we use this algorithm when comparing the array of known\_for or casts, to compute the similarity of two arrays.

**d) N-Grams:** Bigrams and Trigrams are used combining with Jaccard Coefficient to compute the similarity of title of movie, etc.

**iii) Transitivity, Exclusive and Functional Dependency:** These three theorems are used to speed up the procedure of pairwise matching by eliminating some the data entries. These entries’ matching can be determined by these theorems without further comparison.

**2) Blocking (Clustering) based on Character:** if the number of data entries in one dataset are too large, since the running time of one run of pairwise matching is O(n2), it will waste lots of time. If we can divide them into K block, the running time of one run can be reduced to O(n2/k).

**i) Blocking movies:** simply use the lowercase of the first character of the movie’s title to block the data into different sub-dataset.

**ii) Blocking actors:** use both the first letter of first name and last name to form a key. (notice that ‘ba’ will be rearranged to ‘ab’ to compare more potential pairs)

**4.4 Evaluation of Entity Resolution**

**1) Precision and Recall:**

**2) Time used**: the real running time of entity resolution is not very slow since we add some additional procedure to speed up the naive pairwise matching.

**5. Data Fusion**

**5.1 Methods and Algorithms of Data Fusion**

acbb

**5.2 Evaluation of Data Fusion**

aadb

**6. Data Portal**

**6.1 Realized Functions**

abcde

**6.2 Database Optimization**

Eradf

**6.3 Portal Testing**

sasdf

**7. Conclusion**

As all the procedures are done properly, we have built a Movie & Actor Portal providing the integrated information from themovie.org, imdb.com and wikipedia.org. Two methods in both entity resolution and data fusion and also some other algorithms are included to make a better data integration. In the final data portal, we provide several useful and interesting functions to get information from the integrated database.

**8. Reference**

1. ISO 639 Language Code List:<https://www.loc.gov/standards/iso639-2/php/code_list.php>

2. Felix Naumannm, "Similarity measures" [DPDC\_12\_Similarity]

3. JENS BLEIHOLDER and FELIX NAUMANN, "Data Fusion", \_ACM Computing Surveys, Vol. 41, No. 1, Article 1\_