CS 799
Master's Research

Report on

Acquiring large datasets for EMS

Rohith Subramanyam
rohithvsm@cs.wisc.edu
with
Prof. AnHai Doan
anhai@cs.wisc.edu
Department of Computer Sciences
University of Wisconson-Madison
Fall 2014

Index

1) Objective

The objective of this project is to acquire large datasets from two different and independent sources, which represent the same real-world entity.

1.1) Long-term goal

The eventual need for these datasets is to help build a capability in EMS to handle large datasets, all of which cannot fit into memory at once. EMS is a data matching system developed here at the University of Wisconsin-Madison by Prof. AnHai Doan and his research group.

1.2) Why 2 datasets?

Since EMS is a data matching system, it is required that these two datasets are about the same real world entity as we are trying to match records across these datasets.

1.3) Independent

It is also necessary that the sources of these two datasets are independent. Or else, we will end up with a lot of exact matches. And, in real world, when you are trying to perform the hard task of data matching, it is highly unlikely that you will find the data of matching entities having exact matching attributes about them in two different sources. This is what makes this task very hard and the subject of a lot of research to build techniques to make this easier.

1.4) Size of the data

Each dataset should have at least 100,000 records. Ideally, we would like to have 1 million records in each dataset. The dataset should also have a rich set of attributes with at least 6-7 attributes representing the data.

2) Sources

The dataset is about movies. The 2 data sources are:

2.1) IMDb (Internet Movie Database)

is an online database of information related to films, television programs, and video games, taking in actors, production crew, fictional characters, biographies, plot summaries, and trivia (taken from Wikipedia).

2.2) OMDb (The Open Movie Database)

is a free web service to obtain movie information, all content and images on the site are contributed and maintained by their users.

3) Attributes

3.1) IMDb

IMDb table has the following 12 attributes:
$ head -1 imdb.csv
imdbid,title,year,genres,director,writer,cast,runtime,country,language,rating,plot

3.2) OMDb

OMDb table has the following 22 attributes:

$ head -1 omdb.csv
id,imdbid,title,year,rating,runtime,genre,released,director,writer,cast,metacritic,imdbRating,imdbVotes,poster,plot,fullPlot,language,country,awards,lastUpdated,type
The imdbid can be used to label the golden data.

4) Dataset

Datasets are in csv format as EMS currently supports importing tables in csv format.

4.1) IMDb

http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/imdb.csv

4.2) OMDb

http://pages.cs.wisc.edu/~rohithvsm/cs799/omdb/omdb.csv

5) Number of records

5.1) IMDb

1.1 million
$ wc -l imdb.csv
1132263 imdb.csv

5.2) OMDb

2.3 million
$ wc -l omdb.csv
2302427 omdb.csv

6) Data acquisition

6.1) IMDb

6.1.1) List of movies

IMDb publishes a subset of the IMDb plain text data files here: http://www.imdb.com/interfaces. Downloaded the movie list from their ftp site.
This list contains all the movies in IMDb. You can take a look at it here: http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/movies.list

6.1.2) Program

Wrote a python program to iterate through all the movies in this file and fetch details of each movie using HTTP APIs made available here under GPL 2 license: http://imdbpy.sourceforge.net/. The program can be found here: http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/imdb_aggregator_py.txt

6.1.2.1) Parallelization

The python program is a parallelized code using python's multiprocessing module to parallelize the computation on the CPU.
I split the IMDb's movie list into 8 equal parts using the unix command:
$ split -l 134228 movies.list movies.list.segment

6.1.2.2) Deduplication

The python program made use of 8 parallel sub-processes with each process working on one split. For each movie title, a HTTP request is made to fetch the attributes of the movie. This returns multiple results matching the words in the movie title. The attributes are fetched for each result. So, it is highly likely that the more than one process would work on the same movie even though it is working with its own split. To avoid this, the processes made use of a shared memory data structure (shared dictionary), inserting the imdbids of movies that each has downloaded so far into it to ensure that these processes do not do redundant work. And, before fetching the attributes for each movie, a lookup is done by each process to check whether the it has already been completed by another process.

6.1.3) Infrastructure

I ran this program on Amazon Web Services (AWS) T2.micro instance, which has the following configuration:

- 1 virtual CPU (High Frequency Intel Xeon Processors operating at 2.5GHz with Turbo up to 3.3GHz)
- 1GB memory

6.1.4) Duration of the program execution

The program ran for duration of 5 days, 20 hours, 7 minutes and 20 seconds

6.1.5) Post-processing the data

6.1.5.1) Merge

I merged the 8 csv files into 1 using the following program I wrote:
http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/merge_csv_py.txt
The individual splits can be found here:
http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/splits/output/

6.1.5.2) Removing duplicates

The csv file had 2 duplicate records even though I used a synchronization mechanism (shared dictionary) among the processes. This is probably due to a race condition or some other oncurrency issues. Identified the duplicate records using the following unix command and removed them manually:
cat imdb.csv | awk -F, '{print $1}' | sort -n | uniq -d

6.1.5.3) Add release year attribute

The release year of the movie is part of the title and is not got as a separate attribute from the HTTP APIs. I wrote a python program to iterate through all the movie titles in the csv file and extract the release year attribute using a regular expression and add it as another comma-separated field in the csv file. The program can be found here:
http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/add_year_py.txt

6.2) OMDb data:

Followed a procedure similar as stated above. Obtained the database dump by writing to the person who maintains the OMDb. I was a bit concerned whether OMDb is simply fetching its data from IMDb. Hence, wanted to write to the owner and confirm that it is not. This is because I wanted to ensure the 2 sources are independent of each other. Data in OMDb is contributed and maintained its users. Please find the mail conversation below:

Date: *December 5, 2014 at 2:26:17 PM CST*
Subject: Re: question about database dump
From: *Brian Fritz <bfritz@fadingsignal.com>*
To: *Rohith Subramanyam <rohithvsm@cs.wisc.edu>*

Hey Rohith,
The monthly dumps contain all the data provided by the API.
It currently contains about 940k movies, the complete dump has over 2 million records of movies + episodes/series etc.
The data comes from various sources none of which are from IMDb directly (due to legal reasons). So depending on how you plan on using this data it should be fine and mostly fall under the fair use act.

On Mon, Dec 1, 2014 at 5:02 AM, Rohith Subramanyam <rohithvsm@cs.wisc.edu> wrote:
Hi,
I'd really appreciate if you could answer my below questions.
Thanks for your time and consideration.

On Mon, Nov 24, 2014 at 1:47 AM, Rohith Subramanyam <rohithvsm@cs.wisc.edu> wrote:
Hi,
I am interested in a database dump of OMDb as I want this data for a research project in my graduate school. Can you please help me answer the following questions:
How many movies does it have?
Is the movie information pulled from IMDb or is it independent from IMDb?

Please help me answer these questions.
—
// rohith
/ Graduate Student*
Department of Computer Sciences
*University of Wisconsin-Madison */*

Fetched the different attributes using https://github.com/dgilland/omdb.py (made available using MIT license), which is also HTTP based. This job was also written using parallelized python code and took a little more than a week to run. I did not measure the exact duration. It was run on the same AWS micro instance.

7) Other sources considered for Movie dataset:

- Netflix
- Rotten Tomatoes
- Google Play Movies
- iTunes

But none of them had enough movies to get a million records.

8) Other datasets considered

- Music database
- Places database

9) Challenges:

9.1) 2 datasets

It is very easy to get large datasets. But, it is very difficult to get large datasets from 2 different data sources representing the same entity.

9.2) Duration of the data acquisition process

Program to aggregate details about >1 million movies is going to take a long duration to run as it translates to at least so many HTTP requests. I initially tried a single-threaded program

and this was fetching the details of around 5-10 movies per minute. At this rate, it would take more 2 months to get the data for a million movies.
I then wrote a multi-threaded python program to perform the same task, and it did around 50 movies per minute. Python's multi-threading is restricted by its GIL (Global Interpreter Lock). At this rate, it would take a couple of weeks.
I then wrote a multiprocess python program, which is truly parallelized as it spawns multiple subprocesses. This was doing an average of around 150 movies/minute.

9.3) Robustness

Since the data access method is HTTP, I had to run this on a machine, which had reliable network connectivity. Otherwise, it would end up with a lot of errors. Hence, I chose Amazon's AWS to run this program even though it was not on very high-end hardware configuration with the AWS account I have.

Considering the program also runs for a long duration, I had to handle any intermittent errors or exceptions that could occur to ensure that such errors do not disrupt the data acquisition process and I had to start all over again. Any error or exceptions that could occur while fetching the movie information is caught and logged to an error log. The error log for each of the 8 process can be found here: http://pages.cs.wisc.edu/~rohithvsm/cs799/imdb/logs/

Thank you.