IMDb

Link analysis using PageRank index

Mehdi Abbassi - *DSE (947511)*

Supervision

– Professor Dario Malchiodi (dario.malchiodi@unimi.it)

The dataset

THE CHOSEN DATASET, AND THE PARTS OF THE LATTER WHICH HAVE BEEN CONSIDERED

IMDb (an abbreviation of Internet Movie Database) is an online database of information related to films, television series, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, ratings, and fan and critical reviews. IMDb began as a fan-operated movie database on the Usenet group "rec.arts.movies" in 1990, and moved to the Web in 1993. It is now owned and operated by IMDb.com, Inc., a subsidiary of Amazon. As of March 2022, the database contained some 10.1 million titles (including television episodes) and 11.5 million person records.[1]

Subsets of IMDb data are available for access to customers for personal and non-commercial use and is subject to owner's terms and conditions. Each dataset is contained in a gzipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. The first line in each file contains headers that describe what is in each column. The available datasets are as follows:[2]

- title.akas.tsv.gz
- title.basics.tsv.gz
- title.crew.tsv.gz
- title.episode.tsv.gz
- title.principals.tsv.gz
- title.ratings.tsv.gz
- name.basics.tsv.gz

Since we will only use 'title.basics.tsv.gz' and 'title.principals.tsv.gz', we introduce their attributes. The 'title.basics.tsv.gz' has the following attributes:

- tconst (string) - alphanumeric unique identifier of the title
- titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)
- primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release
- originalTitle (string) - original title, in the original language
- isAdult (boolean) - 0: non-adult title; 1: adult title
- startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year
- endYear (YYYY) – TV Series end year. '\N' for all other title types
- runtimeMinutes – primary runtime of the title, in minutes
- genres (string array) – includes up to three genres associated with the title

And 'title.principals.tsv.gz' has the followings:

- tconst (string) - alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- nconst (string) - alphanumeric unique identifier of the name/person
- category (string) - the category of job that person was in
- job (string) - the specific job title if applicable, else '\N'
- characters (string) - the name of the character played if applicable, else '\N'

The structure of the data

Here is the first five tuples of `name.basics.tsv`:

```
+---------+---------------+---------+---------+--------------------+------------------+
|   nconst|    primaryName|birthYear|deathYear|   primaryProfession|    knownForTitles|
+---------+---------------+---------+---------+--------------------+------------------+
|nm0000001|   Fred Astaire|     1899|     1987|soundtrack,actor,...|tt0050419,tt00...|
|nm0000002|  Lauren Bacall|     1924|     2014|  actress,soundtrack|tt0117057,tt00...|
|nm0000003|Brigitte Bardot|     1934|       \N|actress,soundtrac...|tt0049189,tt00...|
|nm0000004|   John Belushi|     1949|     1982|actor,writer,soun...|tt0078723,tt00...|
|nm0000005| Ingmar Bergman|     1918|     2007|writer,director,a...|tt0050986,tt00...|
+---------+---------------+---------+---------+--------------------+------------------+
```

Here is the first five tuples of `title.akas.tsv`:

```
+---------+--------+----------+------+--------+-----------+----------+---------------+
|  titleId|ordering|     title|region|language|      types|attributes|isOriginalTitle|
+---------+--------+----------+------+--------+-----------+----------+---------------+
|tt0000001|       1|Carmencita|    HU|      \N|imdbDisplay|        \N|              0|
|tt0000001|       2|Carmencita|    DE|      \N|         \N|        \N|              0|
|tt0000001|       3|Carmencita|    FR|      \N|         \N|        \N|              0|
|tt0000001|       4|Carmencita|    US|      \N|         \N|        \N|              0|
|tt0000001|       5|Carmencita|    \N|      \N|   original|        \N|              1|
+---------+--------+----------+------+--------+-----------+----------+---------------+
```

Here is the first five tuples of `title.basics.tsv`:

```
+---------+---------+------------+-------------+-------+---------+-------+--------------+-------+
|   tconst|titleType|primaryTitle|originalTitle|isAdult|startYear|endYear|runtimeMinutes| genres|
+---------+---------+--- --------+-------------+-------+---------+-------+--------------+-------+
|tt0002402|    short|The Old G...|The Old Gu...|      0|     1912|     \N|            20|  Short|
|tt0012256|    movie|Headin’ N...|Headin’ North|      0|     1921|     \N|            \N|     \N|
|tt0013347|    movie| Lyda Ssanin|  Lyda Ssanin|      0|     1923|     \N|            \N|  Drama|
|tt0013810|    movie|Ödets red...|Ödets redskap|      0|     1922|     \N|            \N|     \N|
|tt0028192|    movie|Roamin’ Wild| Roamin’ Wild|      0|     1936|     \N|            58|Western|
+---------+---------+------------+-------------+-------+---------+-------+--------------+-------+
```

Here is the first five tuples of `title.principals.tsv`:

```
+---------+--------+---------+---------------+--------------------+--------------+
|   tconst|ordering|   nconst|       category|                 job|    characters|
+---------+--------+---------+---------------+--------------------+--------------+
|tt0000001|       1|nm1588970|           self|                  \N|   ["Herself"]|
|tt0000001|       3|nm0374658|cinematographer|director of photo...|            \N|
|tt0000002|       2|nm1335271|       composer|                  \N|            \N|
|tt0000005|       1|nm0443482|          actor|                  \N|["Blacksmith"]|
|tt0000005|       2|nm0653042|          actor|                  \N| ["Assistant"]|
+---------+--------+---------+---------------+--------------------+--------------+
```

Here is the first five tuples of `title.ratings.tsv`:

```
+---------+-------------+--------+
|   tconst|averageRating|numVotes|
+---------+-------------+--------+
|tt0000001|          5.6|    1550|
|tt0000002|          6.1|     186|
|tt0000003|          6.5|    1207|
|tt0000004|          6.2|     113|
|tt0000005|          6.1|    1934|
+---------+-------------+--------+
```

Pre-processing

The projects requires to implement a system that ranks nodes in a graph using the PageRank index. Nodes in the graph will identify movies, and an edge will link two nodes if the corresponding movies share at least an actor/actress. So we don't distinguish between actor and actress; besides, we are only interested in movies and not documentaries, series, etc. So we join `title.basics.tsv` and `title.principals.tsv` and keep only the attributes of `category` and `titleType` that we are interested in.

```
  SELECT tp.nconst, tb.tconst
  FROM titlebasics as tb
  INNER JOIN titleprincipals as tp
  ON tp.tconst = tb.tconst
  WHERE (category = 'actor' or category = 'actress')
         and
        (tb.titleType = 'movie' or tb.titleType = 'tvMovie');
```

And here is a snapshot of the result. `nconst` represents identifier of the actors/actresses and `tconst` represents the identifier of the movies. So the result is the correspondence between the movies and the actors/actresses who played a role in that movie.

```
+---------+---------+
|   nconst|   tconst|
+---------+---------+
|nm1010955|tt0000335|
|nm1012612|tt0000335|
|nm1011210|tt0000335|
|nm1012621|tt0000335|
|nm0675239|tt0000335|
|nm0675260|tt0000335|
|nm0624446|tt0000630|
|nm0097421|tt0000676|
|nm0140054|tt0000676|
|nm0691995|tt0000793|
|nm5289318|tt0000862|
|nm5289829|tt0000862|
|nm0264569|tt0000862|
|nm0386036|tt0000862|
|nm0511080|tt0000862|
|nm5188470|tt0000862|
|nm0034453|tt0000941|
|nm0140054|tt0000941|
|nm0243918|tt0000941|
|nm0294022|tt0000941|
+---------+---------+
```

Now we need to group the movies an actor/actress played.

```
rdd = rdd.map(lambda x: (x[0], [x[1]])).reduceByKey(lambda x, y: x+y)
```

We are interested in actors/actresses that participated in more than one movies, because it means that it creates an edge between two movies in our network.

```
rdd = rdd.filter(lambda x: len(x[1]) > 1)
```

Then we keep only the list of movies that share a specific actor/actress

```
rdd = rdd.map(lambda x: x[1])
```

Furthermore, make all the possible pairs of movies who share an actor/actress. Note that we want to have a bidirectional network, so ‘combinations‘ instead of ‘permutations‘ is not a good idea. Also note, since many movies share more than one actor/actress removing duplicates using ‘.distinct()‘ is necessary.

```
rdd = rdd.flatMap(lambda x:itertools.permutations(x, 2)).distinct()
```

Now we have a graph where the vertices are the movies and the edges show if the vertices (i.e., movies) share any actors/actresses. The next step is to implement the PageRank indexing algorithm.

The algorithm

First we define a function to calculate how much a vertex contributes from its rank to the neighbouring vertices. As we know in our case the graph is bidirectional and the number of out-neighbours is equal to the number of total neighbours. And each vertex contributes all of its rank equally to all its neighbours, as the algorithm requires.

```
def cal_contribs(outneighbors, rank):
    """
    calculates vertex contributions to
    the rank of other out-neighbours.
    """

    num_outneighbors = len(outneighbors)
    for outneighbor in outneighbors:
        yield (outneighbor, rank / num_outneighbors)
```

Then we create an adjacency list by grouping all edges by their vertices.

```
adjacency_list = rdd.groupByKey().cache()
```

And them we set a initial rank for the vertices. We set the initial rank to one because we experimented that the results when we initialised with one over number of the vertices is the same as when we start with one; besides, one over number of vertices is a very small number.

```
ranks = adjacency_list.map(lambda outneighbors: (outneighbors[0], 1.0))
```

Here comes the core of the implementation. We set a for loop that iterates `num_iters` times, say 10 times, and on each iteration we calculates vertices' contributions to the rank of their neighbours and then calculate the new ranks of the vertices based on the sum of the ranks of their in-neighbours (that in our case are equal to the number of total neighbours) modified by the `damping` (damping factor). It worth noting that since checking the convergence is costly, because we need to collect the results each and every time in order to be able to check the convergence; so we just iterate certain number of the times.

```
from operator import add

for iteration in range(num_iters):
  # vnr stands for vertex_neighbors_rank
  contribs = adjacency_list.join(ranks).flatMap(lambda vnr: cal_contribs(vnr[1][0], vnr[1][1]))

  # re-calculates vertex ranks based on in-neighbours' contributions
  ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank * damping + (1 - damping))
```

Last but not least, we sort the result in a reverse order (starting from the vertex with the highest value). And then collect the results.

```
ranks_sorted = ranks.sortBy(lambda x: -x[1]).collect()
```

Scaling up

Since our implementation has no element causing exponential growth in computation with respect to increase of the data, there is no scale up problem. The computation increases linearly with respect to the size of the data.

Experiments and Results

A DESCRIPTION OF THE EXPERIMENTS COMMENTS AND DISCUSSION ON THE EXPERIMENTAL RESULTS There are 6,326,545 records in `title_basics` and out of that only 536,248 or 8 percent are `movie`. Also there are 36,499,704 'jobs' in our IMDb dataset; one person can have multiple jobs (actress, producer, composer, etc.) even in one single movie. 14,830,233 out of 36,499,704 job are actor/actress, this is roughly 41 percent.

Another interesting point is that always executing the first iteration is very costly compared to the next iterations! For example if the execution of the first iteration takes 14 minutes in a machines, the execution of the second, third, fourth... iterations each takes around two minutes on the same machine.

Here is the first 10 movies with highest PageRank index in our graph and implementation. Note that we just considered a shared actor/actress and this does not mean that the results should be famous, high-quality, or any other specific movies.

1. Dawn of 5 Evils (tt5918332)
2. Exposure (tt11075800)
3. Expel the Wicked (tt4581992)

4. Devotion (tt4167532)
5. Mission: The Prophet (tt3441366)
6. Beyond the Game (tt3546678)
7. The Immortal Wars (tt5259598)
8. Spreading Darkness (tt1791611)
9. Terror Toons 4 (tt4130510)
10. Hunting Season (tt4946336)

Declaration of originality

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

[1] Wikipedia contributors, "Imdb — Wikipedia, the free encyclopedia," 2022, accessed: 10-December-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=IMDb&oldid=1123667937

[2] "Imdb datasets," https://www.imdb.com/interfaces/, accessed: 10-December-2022.