Universita' degli Studi di Milano
Department of Economics, Management and Quantitative Methods

Link Analysis on an IMDb Dataset

Fabio Caironi

January 10, 2022

Abstract

This lab report describes an application of the PageRank algorithm to a large network of actors playing in the same movies. The paradigm of MapReduce is employed for distributed computing of the PageRank through the programming language Python 3.

1 Introduction and Dataset

Finding popular nodes in large graphs has been a popular topic itself ever since Google launched its search engine with the PageRank algorithm. The PageRank can be applied to a variety of contexts, besides web pages, for example for finding people that are most popular or engaged in a network of personal or professional relationships. In this experiment, we've applied the PageRank to a graph representing actors who have met in at least one movie, i.e. nodes are actors and edges link two actors if they've played together at least once. What we expect to obtain is a ranking reflecting the level of "engagement" of an actor with the actors' community and the number and quality of collaborations. Furthermore, because our dataset was large and the PageRank calculation is usually addressed to distributed computing, a MapReduce framework has been adopted with the usage of Apache Spark.
Our experiment has been conducted on a IMDb's dataset available on Kaggle, containing a certain amount of actors' records as well as movies'. Precisely, the dataset has the following tables:

- `title.basics`. Basic information about movies, like title and year, and a movie identifier "`tconst`".
- `title.principals`. Associations between movies and persons, respectively through the identifiers `tconst` and `nconst`. The role of the actor in the movie is also specified. This dataset will be cleaned and renamed to `movie_person`.
- `name.basics`. Basic information about persons, like name, surname and primary profession. This dataset will be cleaned and renamed to `person`.
- `title.ratings`. Average movie ratings.

Computations for this project required high memory ($>$ 15 GB) and a powerful CPU, besides several gigabytes of storage: a Google Cloud's n1-standard-16 virtual instance with 60 GB of RAM and 16 vCPUs has been provisioned and used.
The programming language used for this project is Python 3, and Apache Spark was employed for

the distributed computation of the PageRank (library `pyspark`). For a better readability and the execution segmentation, the experiment has been divided in three Jupyter Notebooks:

- **IMBD__preprocessing.jpynb** contains data acquisition, filtering, matching, re-indexing and graph creation.
- **IMDB__PageRank.jpynb** contains PySpark configuration and the PageRank computation.
- **IMDB__results.jpynb** contains result visualisation and some closing insights.

1.1 Preprocessing and connection matrix

In order to generate the connection matrix described in the paragraph above, the raw datasets had to be cleaned and prepared (see **IMBD__preprocessing.jpynb**: *Filter - Tables match - Re-indexing and resize*) for the link creation algorithm (*Links creation*). The latter was implemented by grouping the `movie_person` table by movie and scanning through a nested for-loop the all possible *sets* of 2 actors playing in the same movie while creating two corresponding arcs. Despite the little improvement of scanning only half of all the possible couples[1], the order of magnitude couldn't be improved and this step resulted in heavy memory usage. To ease the burden on the machine, each unused variable during the link creation was stored and removed from main memory, and reloaded later. The final list `links` contains tuples in the form `(i,j)` where `i` and `j` are integer numbers identifying two actors that played together in at least one movie.
Finally, the connection/transition matrix M was created from the `links` variable and stored efficiently as a list of 3-tuples consisting of, respectively, the row index i representing the target node, the column index j representing the starting node, and the inverse outdegree of node j:

$$M = \left\{ (i, j, o_j) \mid (i, j) \in \texttt{links} \ , \ o_j = \frac{1}{\#\{k : (k, j) \in \texttt{links}\}} \right\}$$

2 PageRank

Our actors graph has dead ends, or actors that seem to have no connection with other peers in any movie. In this case, the connection matrix is said "substochastic" and the process can't longer be considered a Markov chain. If we run the standard PageRank algorithm in presence of dead ends, eventually the distribution will be biased towards such dead ends. A similar bias usually happens with spider traps: however, in the presence of edges instead of arcs and in the absence of one-loops, spider traps won't simply exist, because if you can reach the cycle then you must be able to leave it. To overcome the presence of dead ends, while removal and recursive PageRank re-calculation could be a solution, we instead opted for tweaking the PageRank algorithm with the so called *taxation* method. With taxation, the natural random path of a surfer can be broken at any step with probability $(1 - \beta)$ and restart from a random node chosen with equiprobability. In mathematical terms, the probability that surfer X winds up at node i at iteration m, or $\mathbb{P}(X^{(m)} = i)$, which is equal to $M \cdot [\mathbb{P}(X^{(m-1)} = j)]_{\{j=1,\ldots,n\}}$ according to the standard PageRank iteration model[2], with taxation becomes:

$$\mathbb{P}(X_{tax}^{(m)} = i) = \beta \cdot \mathbb{P}(X^{(m)} = i) + \frac{1 - \beta}{n} \tag{1}$$

where n is the total number of nodes. In our experiment, we've set $\beta = 0.9$.

[1]We scanned $\frac{m(m-1)}{2}$ couples instead of m^2 for each movie, their ratio being asymptotically $\frac{1}{2}$.
[2]It must be remarked that when matrix M is substochastic, so is the variable $X^{(m)}$, thus $\mathbb{P}(X^{(m)} = i)$ is no more a probability distribution, but can be approximated as such.

2.1 MapReduce and PageRank

For an efficient computation of the PageRank, the distributed paradigm of MapReduce was used (see **IMDB__PageRank.jpynb** - *PageRank*). Typically, for large distribution vectors ($\sim 10^9$), which can't fit into a single node's memory, variations of the standard MapReduce algorithm are used, like stripes or blocks partitions; however, since our distribution vector weighted only 13.4MB (1,675,604 floating-point elements, that in Python are roughly of 8 bytes each), we've proceeded with the standard algorithm, of which an explanation follows.
Each iteration will update the PageRank $\mathbf{v}$ through formula 1, starting from the uniform distribution $\mathbf{v}_0 = \frac{\mathbf{e}}{n}$. More precisely, the iteration step is:

$$\mathbf{v'} = \beta M\mathbf{v} + (1-\beta)\frac{\mathbf{e}}{n} \tag{2}$$

The principal gain of MapReduce comes in the matrix-vector multiplication of the sparse transition matrix M by the PageRank $\mathbf{v}$:

- A first map action maps all non-zero elements of the transition matrix in a key-value store, where the key is the row index i and the value is the contribution of that matrix element to the i-th component of $\mathbf{v'}$ in the product $\beta M\mathbf{v}$:

$$\begin{aligned} map_1 : M^{RDD} &\longrightarrow A \subset \mathbb{N}\times\mathbb{R} \\ m_{ij} &\longmapsto (i, \beta m_{ij} v_j) \end{aligned} \tag{3}$$

- It follows a reduce step where all the addends with the same row key are reduced by commutative summation:

$$\begin{aligned} red : \mathcal{P}(A) &\longrightarrow B \subset \mathbb{N}\times\mathbb{R} \\ \big[(i, c_j)\big]_j &\longmapsto \Big(i, \sum_j c_j\Big) \end{aligned} \tag{4}$$

- Finally, teleportation is added through a last map step:

$$\begin{aligned} map_2 : B &\longrightarrow C \subset \mathbb{N}\times\mathbb{R} \\ (i, c) &\longmapsto \Big(i, c + \frac{1-\beta}{n}\Big) \end{aligned} \tag{5}$$

The result is consistent with 2, indeed the i-th element of $\mathbf{v'}$ can be obtained through composition of the three steps:

$$\begin{aligned} map_1 \circ red \circ map_2 : M^{RDD} &\longrightarrow C \subset \mathbb{N}\times\mathbb{R} \\ m_{ij} &\longmapsto \Big(i\ ,\ \sum_j \beta m_{ij} v_j + \frac{1-\beta}{n}\Big) \\ &= \Big(i\ ,\ \beta\sum_j m_{ij} v_j + \frac{1-\beta}{n}\Big) \end{aligned} \tag{6}$$

A chunk of code has been added after the map-reduce-map procedure because of the following problem: there were some nodes without incoming arcs and the corresponding row index was missing from the list representation of the connection matrix; as a consequence, those nodes were not considered in either map and reduce steps. Despite zero-rows don't bring any contribution to the corresponding node's PageRank, with teleportation *all* nodes should be equally affected by the summation of $\frac{1-\beta}{n}$. Therefore, at each round, our code scanned all the resulting keys, detected the missing ones and added teleportation to the PageRank of the corresponding nodes.

3 Results

The pre-processed database contained 1,675,604 actors and 14,059,984 movie-actor associations, with the actors table (`person`) weighing 83.8 MB and the movie-actor (`movie_person`) table weighing 339.2 MB. The connection matrix creation required more than 15 GB of RAM usage and was stored in a file of 568.9 MB, which is only 0.0025% of the size of a $n \times n$ matrix with $n =$1,675,604. The Spark implementation of the PageRank algorithm was run starting from `n=1000` partitions of the connection matrix's RDD. Each iteration first compared the old page rank vector with the current one through the $l2$ distance: when a tolerance of 10^{-10} was reached, the execution was stopped. After 70 iterations, split in different computing rounds by using checkpoint values, we obtained an estimate of the PageRank meeting that tolerance.

3.1 Visualization

In order to make such a long PageRank human-readable, we only inspected the top-10 actors and their connections and films (see **IMDB_results.jpynb** - *Data visualization*).
The highest PageRank went to Christian Bobet, with a value of $6 \cdot 10^{-5}$, which is 100.5 times higher than the equiprobability. Bobet played in 355 movies with other 1250 actors. The second top-ranked actor had a PageRank of $4 \cdot 10^{-5}$, having played in 1008 movies with 2090 actors, while the third one played in 442 movies with 1165 actors. If we look for example at the same metrics for the actor ranked 1000th, we see a decline in the number of connections: he played in 595 movies with only 542 actors. These figures tell us that despite the PageRank is overall sensitive to the volume of connections, it does care more whether those connections are "strong", like in the case of the first and second top-ranked actors. Some of the top-ranked actors also "played" together, and if the reader has the patience to go a couple of lines below will also understand why. Finally, having played in a lot of movies with a lot of other actors doesn't make you a good actor: the top-rated movie starring Christian Bobet had a rating of only 5.7/10!

Unfortunately, some of the aforementioned top-ranked actors in this experiment are coming from the adult movie industry. The writer disclaims any responsibility from finding such results and declines to draw further conclusions, even though it could prove funny. Besides, the reader can replicate the experiment on another dataset from IMDb or match actors with movies having the `is.adult` attribute equal to 0 in the original table.

References

Mining of Massive Datasets. Leskovec, Jure and Rajaraman, Anand and Ullman, Jeffrey David, 2014, Cambridge University Press, 431, 28.

Statement of IP