Link analysis on the Yelp dataset of user reviews

Luigi Foscari

Algorithms for massive dataset

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.

1 INTRODUCTION

This project explores the use of various ranking algorithms on a graph induced from the *Yelp Dataset* available on Kaggle[1], containing information on the reviews published on the homonymous website about businesses across 8 metropolitan areas in the USA and Canada. The data coincides with version 4 of the dataset published in 2022 and downloaded on the 31th of May, during the analysis both the `reviews` and the `user` tables were used, the storage and analysis was conducted using the Spark engine [Zah+16].

The dataset used contain information on the reviews and on the users, the former was used to build the graph (see section 2) and contains the review author and the business to which the review is targeted, both as string identifiers, the latter for specific examples of topic-sensitive PageRank (see section 3.3).

2 GRAPH

To build the graph from the dataset I connected two users when they both reviewed the same business, the resulting graph is symmetric and weightless, making the adjacency matrix symmetric and binary $M \in \{0,1\}^{n \times n}$, where for $i \neq j$ we can define $m_{ij} = 1$ if at least one business was reviewed by both user i and user j, 0 otherwise. For simplicity I removed self-loops, making the diagonal of M zero. Upon download the `reviews` dataset contains a list of reviews with an associated user and business id, to compute the graph the first step was to self-join the reviews Spark Dataframe on the business id, isolating the two resulting users, after removing the other columns and mapping users to integers what is left is a list of arcs. The mapping to integers was achieved with a simple dictionary, but a more sophisticated solution based on minimal perfect hashing function could be implemented. The matrix M is likely very sparse, to optimize space consumption I mapped the arcs into `SparseVector` objects and then to a `IndexedRowMatrix`. During the analysis different views of matrix M are required, these alternatives were computed before-hand, define $\overline{M}$ as M row-wise normalized in L_1 norm and $\overline{M}^\top$ its transpose[2].

3 RANKINGS

The following is the list of the implemented ranking algorithms.

[1] https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset

[2] The graph on which the implementations work is symmetric, therefore $M^\top = M$, but in general this is not the case and the goal is for the implementations to be applicable to every graph.

3.1 *Degree*

A trivial ranking algorithm ranks each node according to their degree, this is a naïve approach which only considers local neighbourhoods without taking into consideration the complete structure of the graph. Being the graph symmetric the degree of a node is defined as the number of arcs associated to each node. to compute this ranking I simply scanned the non-zero entries of the matrix and incremented the counter associated to the considered user, this approach scales well because it can be performed in parallel on an arbitrary number of partitions of the dataset due to how the counters are implemented, for every partition a local process counts the degree of the local nodes and then the results are joined together. In Google Colab the machines have only 2 cores available, therefore the performance optimization is difficult to show experimentally.

3.2 *PageRank*

PageRank is a spectral centrality proposed by [BP98] initially in the context of web graphs [Gle15] which leverages the whole structure of the graph considering it as a collection of states and transitions instead of pages and links, starting from any state PageRank tries to quantify the likelihood of *surfing* to any other one. In practice we want to find a stationary configuration of this state machine, to achieve this we simulate the machine at work, every time distributing the probability of being in a particular state to all its neighbours according to the respective transition probability; at any point in time t we can define the next value of PageRank for a node j as

$$v(t+1)_j = \sum_{k=1}^{n} v(t)_k m_{jk} \tag{1}$$

where m_{jk} is an element of the transition matrix $\overline{M}^\top$ and represents the arc from k to j and can either be 0 if is absent or a value between 0 and 1; at each step we distribute part of the rank from the neighbour nodes of j to j itself, this process can be rewritten as repeated matrix multiplications of $\overline{M}^\top$ with a rank vector until convergence, which is exactly the power method to compute the principal eigenvalue of $\overline{M}^\top$. It can be proven that if the matrix is column-wise stochastic the power method converges, and that is why we are using $\overline{M}^\top$ and not M or $\overline{M}$, where if a column is non-zero, it defines a probability distributions. In practice the implementation stops refining the rank until either certain number of iterations has been surpassed or a precision threshold is reached (in the code these values are 100 and 10^{-5} respectively). The problem of zero columns happens in graphs where not every node has outgoing arcs, in our case this is impossible because of how the graph is constructed, but for sake of generality this problem was considered, it can be solved in different ways:

- Remove the zero columns, thus removing the corresponding nodes, and define an ad-hoc rank for those nodes after having computed the rank (for example perform one PageRank step only on these nodes). This solution can be problematic because it changes the topology of the graph during computation.
- Add self-loops to these nodes, in practice setting $m_{ii} = 1$ for the node i, this causes rank to accumulate on these *spider trap* nodes.
- Consider another graph and at every iteration of the algorithm the rank can either be distributed to a node who is a neighbour of the source in the original graph or in this new one, then make this new graph a clique and define a probability or *damping factor* β of choosing neighbours from the original or the new graph.

The last solution is the most commonly used under the name of *teleportation* or *taxation method* and has proven to work very well in practice, at every iteration of the algorithm each surfer throws a β-weighted coin and in one case chooses a neighbour from the non-zero values from the right column of $\overline{M}^\top$, otherwise picks a node from the network uniformly at random. In practice we can rewrite the formula 1 as follows:

$$v(t+1)_j = \beta\left(\overline{M}^\top v(t)\right)_j + (1-\beta)\frac{1}{n} \qquad (2)$$

The implementation performs this computation with a couple of optimizations:

- The quantity $(1-\beta)\frac{1}{n}$ is computed in advance and added to the result of the iteration every time.
- The matrix-vector product $\overline{M}^\top v(t)$ is performed in Spark, where $v(t)$ is a Numpy [Har+20] array and $\overline{M}^\top$ is a `IndexedRowMatrix` divided into multiple partitions for parallel computation.
- The rows of $\overline{M}^\top$ are stored in memory as `SparseVector` for optimal space consumption.

To check if the precision threshold has been reached the implementation computes the euclidean distance between each refinement of the rank vector $v(t)$ and $v(t+1)$.

3.3 *Topic-sensitive PageRank*

Whether we are working with web pages or review authors, the random surfing solution works well in practice in helping the PageRank algorithm in evaluating the global topology of the graph, but it fails to consider the semantic of the nodes themselves. Topic-sensitive PageRank [Hav02] at its core changes the teleporting technique we saw in PageRank to an arbitrary (non-uniform) probability distribution on the nodes of the graph. Instead of computing a single ranking to apply to all the users, many different rankings are computed with different targets, the granularity can be chosen at will, for example it could be possible to compute a PageRank vector for every single user based on their preferred web pages (inferred for example by the search history), otherwise the users can be clustered according to different interests or social groups. In the context of Yelp reviews the implementation computes three different rankings by selecting users in the top 20% according to how funny, useful and cool their reviews are, this information is contained in the `user` dataset. After having selected the interesting users the next step is to build a preference vector $e \in \{0,1\}^n$ where for each interesting user i, $e_i = 1$ and 0 otherwise and ultimately normalize this preference vector in L_1 norm, the resulting formula is a slightly modified version of 2:

$$v(t+1) = \beta\left(\overline{M}^\top v(t)\right) + (1-\beta)e \qquad (3)$$

From a technical point of view the implementation of this ranking algorithm is almost identical to PageRank, with the modification that being e constant it can be broadcast to all the Spark nodes to be used for computation without performing repeated copies. Like for the degree in this case the process of building the preference vectors is achieved thanks to local counters merged in the end, making the implementation scalable with the partitions.

3.4 *HITS*

Hyperlink-induced topic search is a ranking algorithms based on the structure of the web, where a page either contains content to be consumed or links to one such page, we call them *authority* and *hub* respectively. Every page is scored as an authority and as a hub, where a good authority is defined as being linked by good hubs and a good hub links to good authorities. This double-recursive definition translates to an elegant formula where an initial value for the authority score is computed and then used to compute the first approximation of the hub score, the next iteration refines the hub score using the authority score, then the roles are switched and so on. The authority score a_i for a page i is defined as the sum of the hub scores of the pages containing a link to i, the hub score h_i for a page i is defines as the sum of the authority scores of the pages linked by i.

$$a_i = \sum_j M_{ij} h_i \qquad h_i = \sum_j \left(M^\top\right)_{ji} a_j \quad (4)$$

To list the predecessors of any node i we can consider the non-zero values of the i-th row of M, to list the successors we consider the i-th row of $M^\top$. An important note to make is that in our case the hub and authority scores coincide, because the graph is symmetric and therefore $M = M^\top$, this can be shown by applying this substitution in 4:

$$a = Mh = MM^\top a = MMa = M^2 a$$
$$h = Ma = MM^\top h = MMh = M^2 h$$

meaning that at every update both scores are scaled the same amount. As stated earlier the implementation ignores the symmetry of the graph to be flexible.

4 RESULTS

Evaluating the quality of a ranking for a graph is a challenging task because there is no standard to which compare the results, to show the importance of single users I decided to isolate the five users with the highest scores for every ranking algorithm in table 1. To evaluate the implementations and the algorithms performance I kept track of the runtime of the algorithms and, in the case of topic-sensitive PageRank, also the processing needed to compute the preference vectors.

Due to hardware limitations on the Google Colab platform the tests were conducted on a subset of the original dataset, in particular I retained only the first 100 000 entries, from which was built a graph with 6 874 736 arcs and 77 604 nodes.

The execution time for each algorithm can be compared in the following table and show very similar performances across the board, interestingly PageRank with random teleporting reaches the threshold faster than the topic-sensitive counterpart, probably due to the lower density of the graph in the latter. All the algorithm with a stopping condition on either a threshold or a fixed number of steps terminated on the threshold.

Ranking	Time (preprocessing)
Degree	0m 41s
PageRank	10m 05s
TsPR (useful)	12m 16s (41.8s)
TsPR (funny)	12m 01s (41.8s)
TsPR (cool)	11m 56s (41.8s)
HITS	08m 09s

The various rankings return values in very different scales, instead of mapping the values in a common space, which could introduce scaling bias[3], and considering that in practice the rankings are useful to determine an order in the set of nodes, I decided to sort the users according to the various computed rankings and analyse the differences in positions for each user using Spearman's rank correlation coefficient, which assesses how well the relationship between two variables can be described using a monotonic function, in practice it's computed exactly as the Pearson correlation coefficient, but the underlying data represents a ranking, not a generic distribution. The results are shown in figure 1 and show a high correlation among all the rankings, it's especially interesting the case of the degree, because it is very close to all the rankings, but can be computed very easily and efficiently. In table 2 it's possible to inspect the ranking distributions, these values were computed by computing 100 buckets and plotting the results in a scatter plot, for the HITS results this solution showed a little amount of data, because the rank stagnates on very low or very high values, to improve the visualisation I decided to remove the outliers. As confirmed in figure 1 we can visually see in table 2 a strong correlation between the rankings, especially the ones based on PageRank. In the HITS rankings there is a clear gap in rank values between roughly 2 and 3, this likely means that the rank is being accumulated in certain regions of the graph, leaving these regions with a very high rank and all the other regions with very low rank.

[3] From the distribution of the rank in HITS it's clear that the values are either very high or very low.

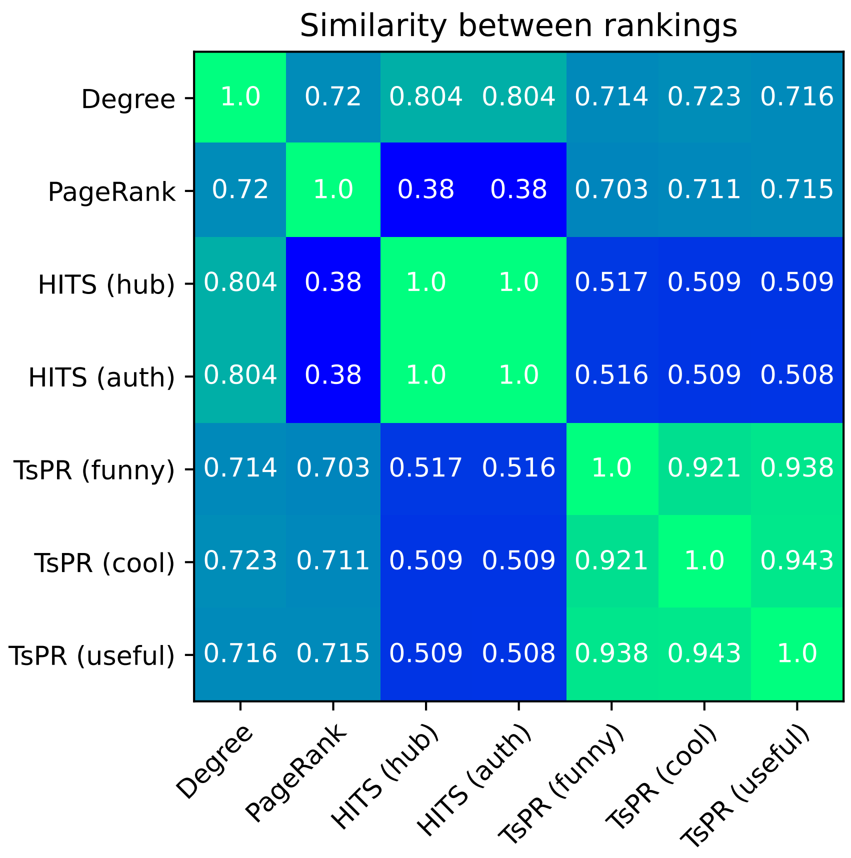

Figure 1: Ranking correlation matrix among the orderings (Spearman).

	Degree	PageRank	HITS (hub)	HITS (auth)	TsPR (funny)	TsPR (cool)	TsPR (useful)
1	CfX4s	_BcWy	LO_YM	LO_YM	_BcWy	_BcWy	_BcWy
2	tNq35	Xw7Zj	hmVE7	hmVE7	1HM81	mzL0z	mzL0z
3	hmVE7	1HM81	tNq35	tNq35	mzL0z	Xw7Zj	1HM81
4	LO_YM	mzL0z	CfX4s	CfX4s	Xw7Zj	1HM81	Xw7Zj
5	xqS2P	ouODo	bdIue	bdIue	Hxx8F	2iS1v	Hxx8F

Table 1: Top 5 heavyhitters for each ranking.

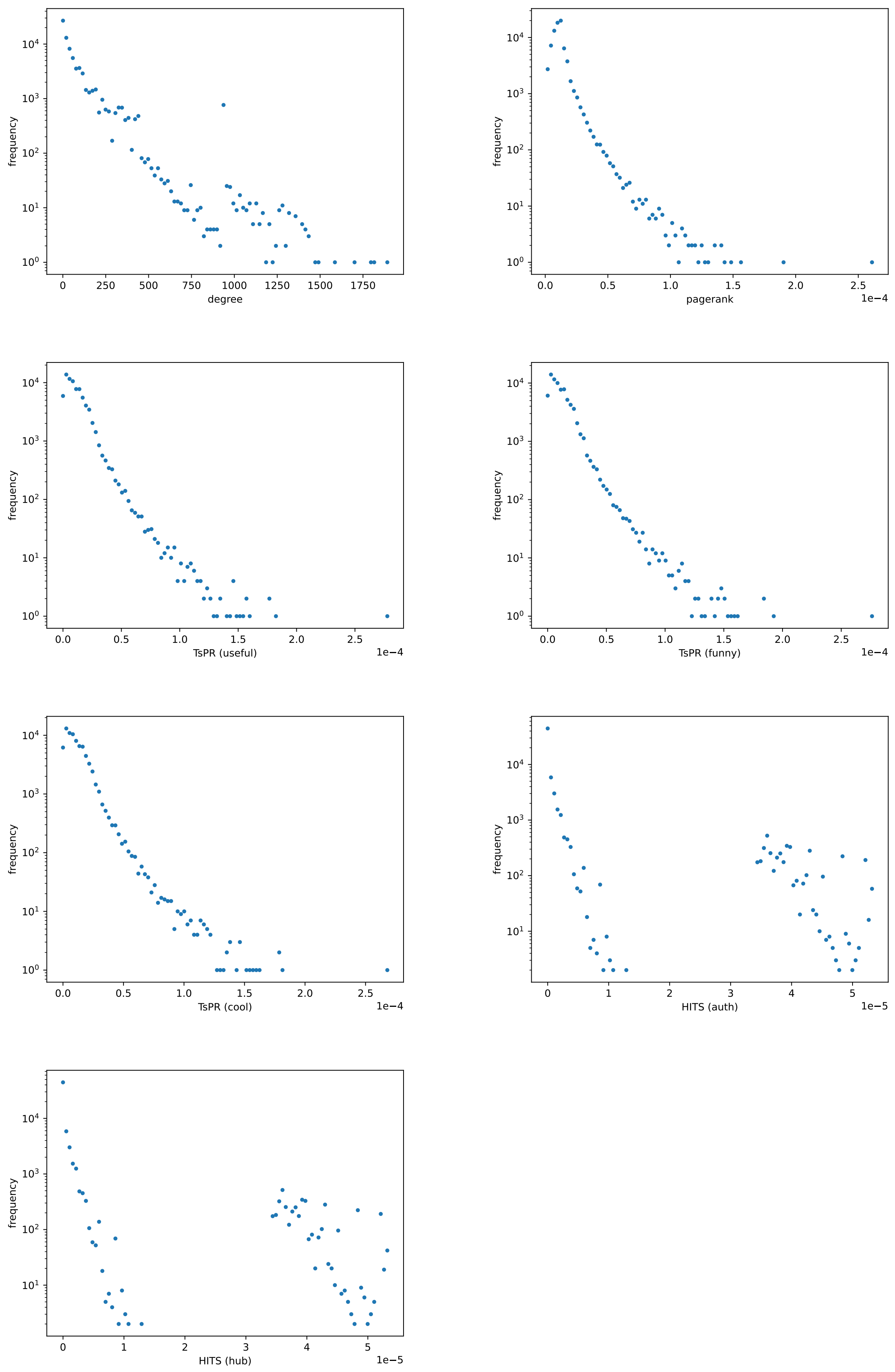

Figure 2: Rank distributions.

REFERENCES

[BP98] Sergey Brin and Lawrence Page. “The anatomy of a large-scale hypertextual Web search engine”. en. In: *Computer Networks and ISDN Systems* 30.1-7 (Apr. 1998), pp. 107–117. ISSN: 01697552. DOI: 10.1016/S0169-7552(98)00110-X. URL: https://linkinghub.elsevier.com/retrieve/pii/S016975529800110X (visited on 06/01/2023).

[Gle15] David F. Gleich. “PageRank Beyond the Web”. en. In: *SIAM Review* 57.3 (Jan. 2015), pp. 321–363. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/140976649. URL: http://epubs.siam.org/doi/10.1137/140976649 (visited on 06/01/2023).

[Har+20] Charles R. Harris et al. “Array programming with NumPy”. en. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-020-2649-2. URL: https://www.nature.com/articles/s41586-020-2649-2 (visited on 06/01/2023).

[Hav02] Taher H. Haveliwala. “Topic-sensitive PageRank”. en. In: *Proceedings of the 11th international conference on World Wide Web*. Honolulu Hawaii USA: ACM, May 2002, pp. 517–526. ISBN: 978-1-58113-449-0. DOI: 10.1145/511446.511513. URL: https://dl.acm.org/doi/10.1145/511446.511513 (visited on 06/01/2023).

[Zah+16] Matei Zaharia et al. “Apache Spark: a unified engine for big data processing”. en. In: *Communications of the ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/2934664. URL: https://dl.acm.org/doi/10.1145/2934664 (visited on 06/01/2023).