Link Analysis of the Amazon US Costumer Review Dataset

March 20, 2023

Author: Sara Gironi

Student number: 976803

Algorithms for Massive Data

Università degli Studi di Milano

Contents

1 Introduction

Nowadays Amazon is leading the global e-commerce market and its products often have thousands of reviews lefts by customers, which are often very useful when deciding which product to purchase. The aim of this project is to create a ranking system that could be used in order to establish the sequence in which the reviews should be shown. The idea is to calculate the PageRank score of the customers based on the reviews they did of an item in common with other customers. This is because it's difficult that spammers (that want to inflate the rating of their products) have reviews in common with others of different products. However, in this way it would be given more importance to customers that left many, but meaningless reviews compared to those who left fewer, but more effective reviews. The PageRank score doesn't give any information of the usefulness of the review left by users, however, on Amazon it is possible for users to rate a review as "useful". So, in addition to the PageRank score the "helpfulness score" is evaluated by summing together the helpful votes by customer, divided for the total number of times a review has been rated as useful for all the customers that have at least an edge. The vector of the "helpfulness score" is added to the PageRank values changing completely the results of the PageRank ordering, advantaging the customers that left fewer, but more useful reviews, compared to who left many, but not really valuable reviews.

The work is developed in the following way: the first section describes the data, the second one describes the theoretical framework behind PageRank algorithm, the teleport variation and the role of MapReduce function. The following sections describe the preprocessing techniques, the exploratory data analysis and the implementation of the algorithm. Finally, the results are compared and discussed. In order to work with large-scale data Python and Apache Spark were used, to distribute computation.

2 The Data

The Amazon US Customer Reviews Dataset is taken from the Kaggle public repository and contains textual data of the reviews and connected metadata that span over 20 years, between 1995 and 2015. It is formed by more than a hundred million of rows subdivided in 37 different categories. The data analyzed in this project belongs to two categories: pc and electronics, for a total of more than 10 million rows. The reasons for the choice of these categories are the followings:

1. The reviews might tend to be more technical and less subjective due to the nature of the product, compared to other categories (such as Books, Music, etc). Therefore, there could be a higher number of "$helpful\_votes$".

2. Electronic and pc categories are highly correlated, so two people buying the same laptop on Amazon might also buy the same mouse which is advertised.

The dataset is composed by the following attributes:

- *customer_id*: unique identifier for each customer.
- *review_id*: unique identifier for each review.
- *product_id*: unique identifier for each product.
- *product_parent*: Random identifier that can be used to aggregate reviews for the same product.
- *product_title*: name of the product
- *product_category*: category to which the product belongs to.
- *star_rating*: The 1-5 star rating of the review.
- $helpful\_votes$: Number of helpful votes.
- *review_date*: The date the review was written.

Other variables such as *marketplace*, *total_votes*, *vine*, *review_headline*, *review_body*, have been dropped in order to make the dataset lighter, as they are not useful for the purpose of this project.

3 Theoretical framework

3.1 PageRank Algorithm

PageRank is the algorithm that made Google prevail over all other search engines. This is because it was the first one to be able to overcome the problem of spammers who had made search almost useless, thanks to its variation of TrustRank and other approaches to detect link spam. The innovation of PageRank is that it doesn't only count the incoming links of a page, but also the importance of the page pointing to it, in order to avoid spam farms.

Google prefers important pages to unimportant pages when deciding which one to show first in response to a search query. The same idea can be applied to many other kinds of data, like, for example, in this project to create a ranking system of Amazon customers. Like webpages the reviews of a product should be shown in order of importance. In this case the nodes are the customers and they are connected by an edge when they have both reviewed the same product. The edges are bidirectional. A customer is more important the more he is linked to other nodes. The PageRank score is calculated through the simulation of the movement of a random surfer around the graph, starting from a random node. At each time step the surfer will move to another node through a randomly chosen outlink. This process is iterated many times, which allows the surfers to converge to a distribution. Customers nodes that will have a large number of surfers are considered more "important" than those that would rarely be visited. The probability for a surfer to move from a node to another one can be represented by the transition matrix M.

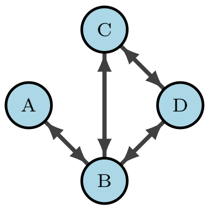
$$M = [m_{ij}]_{nxn}$$

It has n rows and columns, as the number of nodes in the graph and it's a stochastic matrix (each column sums up to 1).

$$m_{ij} = \begin{cases} \frac{1}{k}, & \text{if } j \to i \\ 0, & \text{otherwise} \end{cases}$$

The element m_{ij} in row i and column j has value $\frac{1}{k}$ if page j has k outgoing links, and one of them is to page i. Otherwise, $m_{ij} = 0$.

Simplified example of the graph with only 4 customers:



Transition matrix M:

$$\begin{array}{cc} \begin{array}{cccc} A & B & C & D \end{array} & \\ \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 \\ 1 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{1}{2} & 0 \end{bmatrix} & \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

Initial vector v_0:

$$\begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}$$

At each discrete time step the probability to move from one node to another node doesn't change, because the structure of the graph is constant over time. Therefore, v_0 is the initial vector that will have $1/n$ values that represents if a random surfer starts at any of the n nodes with equal probability. After one step, the distribution of the surfer will be Mv_0, after two steps it will be $M(Mv_0) = M^2v_0$, and so on. The limit is reached when multiplying the distribution by M another time does not change the distribution more than a certain threshold. In fact, because M is stochastic, v is the principal eigenvector (its associated eigenvalue is the largest of all eigenvalues). Note also that, because M is stochastic, the eigenvalue associated with the principal eigenvector is 1. However, the convergence is correct provided that two conditions are met:

1. The graph is strongly connected;
2. There are no dead ends

Anyway, it's very difficult in real life to find a strongly connected graph without dead ends. The graph obtained from the US Amazon Customer Reviews Dataset makes no exception. It doesn't have dead ends because all the arcs are bidirectional, however the graph is not connected. This means that there are spider traps. A spider trap is a set of nodes with no dead ends, but without the possibility for the surfer to reach other nodes than those in the trap. In this sense, once a random surfer enters a spider trap, it can never leave. Once a random surfer enters a spider trap can never leave. To avoid this problem a variation of PageRank has been introduced.

3.2 Teleport variation

The solution consists of modifying the calculation of PageRank by allowing each random surfer a small probability of being teleported to a random page, rather than following an out-link from their current page. This is done by introducing a new element to the matrix vector multiplication:

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

where usually $\beta = 0.85$ The term $\beta M v$ represents the case where, with probability β, the random surfer decides to follow an out-link from their node they are in. The term $(1-\beta)\frac{1}{n}$ is a vector with probability $(1-\beta)$ to be teleported at a random node. This allows to redistribute surfers that are stuck into spider traps.

3.3 MapReduce

The application of PageRank over such a massive dataset wouldn't be possible without the storage of the data into a Distributed File System, such as Hadoop. In order to work with HDFS Map and Reduce functions are used. The data is divided in chuncks, that are stored in different machines that run in parallel. Instead of storing the transition matrix, which is highly sparse, in matrix form, each entry is stored as a triple (i, j, m_{ij}). Applying the Map Function to the triples will turn them into a sequence of key-value pairs. The way key-value pairs are produced from the input data is determined by the code written by the user for the Map function. In this case the output is $(i, m_i v_j)$, where $m_i v_j$ is given by the multiplication between m_{ij} and the entry v_0 corresponding to j. The shuffling phase is when the key-value pairs from each Map task are collected by a master controller and sorted by key. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task. Finally, the Reduce function will have as input $(i, [m_{iv1}, m_{iv2}, m_{iv3}...])$ and will output (i, s), where s is the sum of all $m_i v_n$ elements which is the same result we would obtain through the matrix vector multiplication Mv. In this way MapReduce enables computation on large-scale data to be performed in a way that is tolerant of hardware failures during the computation.

4 Pre-processing techniques

The first step to pre-process the data is to import it as a Spark dataframe, which allows parallel computation and multiple-nodes data partition. The categories of pc and electronic are joined by union for a total of more than 10 million rows. Null values and duplicates are eliminated and type of same variables is cast from string to integer. I also check for anomalies, like in the case if $star\_rating$ has values bigger than 5 or $helpful\_votes$ is bigger than $total\_votes$. No anomalies are detected. After completion of these steps, the dataset is consistent and balanced. It is ready to proceed with sampling and the implementation of the algorithms. For the application of the algorithm just a fraction of the entire dataset is considered, otherwise the computation would be too time consuming. 50000 rows are sampled at random.

5 Exploratory Data Analysis

From an initial exploratory analysis of the data it is possible to make some observations:

- Among the 50000 customers on average only 1.01 reviews are left on the same product. This could be explained by the fact that the sample is very restricted compared to the whole dataset. Also the data is collected over two decades, between 1995-2015, so some products after some years might not be on sale anymore.
- The highest number of reviews left by the same customer is 6.
- The most reviewed item has 111 reviews.
- The review that was more useful has 2431 helpful votes. This means it was useful for thousands of other customers and this data shouldn't be neglected in the creation of the ordering. For this reason it is going to be incorporated in the "helpfulness score".
- The highest numbers of reviews are left in January of years 2014 and 2015. The possible explanation is that Christmas is the moment where most products are purchased and after few days or weeks trying the product customers leave the reviews on Amazon.

6 Implementation of PageRank

The PageRank algorithm is implemented from scratch. The first step is to create a graph where the nodes are formed by customers that are connected by edges when they have reviewed the same product. To do that I first create a pyspark dataframe that grouped customers by product, then two different methods are implemented to create the graph:

1. The first one involves the User Defined Function of Pyspark, which allows to apply a function directly on a pyspark dataframe. In this way I am able to obtain a dataframe with source and destination nodes of each edge. To build the graph with Networkx, the pyspark dataframe must be transformed into a Pandas dataframe and then into a list to iterate on it. After that, by creating a dictionary a numeric index is assigned to each node, because integers are less computationally expensive to deal with compared to string.

2. The second method uses only for loops to obtain the same exact result.

The total number of nodes is 26968 and the total number of edges is 118183, which means that on average each node has around 4.38 edges. The graph obtained is unconnected, this implies the presence of some spider traps. Both methods are timed. The time difference is quite significant (4.3 second for for loops against 245 ms of udf()). When scaling-up the computation to the whole dataset with millions of rows the inefficiency of for loops wouldn't allow the computation in an acceptable time range.

The next phase consist of creating the transition matrix. To obtain it I start by creating an adjacency list for each node that is stored in the adjacency dictionary which has indexes, instead of the customer id values. An empty list is created to store the transition probabilities. So, for each pair of adjacent nodes a tuple with 3 values is appended that correspond to (i, j, m_{ij}) described in the theoretical framework.
The transition matrix is now transformed into a RDD and ideally distributed over different machines where tasks could be performed in parallel. Parallelization and caching guarantee a faster and more efficient computation. The transition matrix is now ready to be fed to the PageRank algorithm.
First the vector v is initialized to describe the uniform distribution over all the nodes of the graph. I set a convergence threshold, that indicates when to stop iterating because the scores are not changing anymore significantly. The threshold is 0.0001 and is calculated by the residual error of each iteration. However, I also set a maximum limit of iterations to 500 to avoid the process to be too time consuming. The iteration of the Map function allows to calculate in parallel the multiplication of m_{ij} for the corresponding v_j value and the output for each node is summed by the Reduce function. Once concluded the iterations the PageRank scores are output. However, these scores are misleading, due to the presence of spider traps. To verify if the score changes with the teleport variation the same procedure is followed again adding the probability of being

teleported at each iteration. The value of β is set to 0.85. The difference between the two versions is calculated and in fact the scores are a bit different.

Pagerank values are all pretty low which is normal as in the exploratory analysis it was already shown that most of the customers had few reviews in common. This is also because the sample chosen is very restricted compared to the size of the original dataset. Furthermore, many nodes have exactly the same score. For this reason I decide to introduce the variable "$helpful\_votes$", to try to diversify between who left a useful review and who didn't, in order to create a better ordering.

6.1 Helpfulness score

The vector with the helpfulness score for each node in the graph is given by:

$$helpfulness\_score(c) = \frac{\sum_{i=1}^{I} h_i}{\sum_{j=1}^{J} h_j}$$

where the numerator is the sum of the number of helpful votes obtained by customer c over all his reviews i divided by the total value of "$helpful\_votes$" for all reviews J given by all customers with at least an edge in the dataset.

The values are then saved in a dictionary and summed by key with the corresponding PageRank score.

7 Results and Conclusions

The final values obtained in the 3 different versions of PageRank are compared in the table below:

PageRank scores	PageRank scores with teleport	PageRank scores with helpfulness
40746147: 0.00015	21360587: 0.00013	52855449: 0.05663
21360587: 0.00015	40746147: 0.00012	12372811: 0.02979
34552600: 0.00011	34552600: 0.00010	41866357: 0.02585
10699969: 0.00011	2444062: 0.00010	45035381: 0.01487
2444062: 0.00011	39135347: 0.00010	15208771: 0.01203

As it's possible to observe comparing the columns the scores of PageRank with or without teleport are very similar and also the top nodes are almost the same, this is because the graph is undirected (edges are bidirectional), while when the helpfulness score is added the ranking changes completely. Some customers left very useful reviews and this increased a lot their score, in fact, node 52855449 which has the highest "helpfulness score" is the customer that got the the highest number of helpful votes. The decision on which method to use depends on what importance one is planning to give to the usefulness of the review itself.

In conclusion, the calculation of the Pagerank score of Amazon users is performed over the Amazon US Customer Reviews Dataset between 1995 and 2015. The PageRank algorithm is implemented with the teleport variation to avoid spider traps. Furthermore, a measure to calculate the helpfulness of each customer is computed and added to the PageRank score. The introduction of this additional value is based on the idea that the feedbacks of users that wrote the most useful reviews should be shown on top. As spammers tend to leave many reviews on their products PageRank scores alone could be not that effective in ordering users, so the "helpfulness score" rewards users who left reviews considered useful by other users. This method assumes that spammers do not inflate their reviews by rating them as useful, because in that case the helpfulness score would be completely misleading. For the future work a way to tackle this issue should be explored.

The code to reproduce this analysis can be found in the following GitHub repository.

"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 us or any other person for assessment on this or any other course of study."