UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE POLITICHE,
ECONOMICHE E SOCIALI

AMAZON US CUSTOMER REVIEWS: LINK ANALYSIS

Algorithms for Massive Datasets

Simona Caruso

A.A. 2022-2023

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.

Contents

1 Introduction

PageRank is an algorithm developed by Google to rank search results based on their relevance, which is determined by the number of direct links pointing to a specific page. If the linking pages themselves are considered relevant, the PageRank score assigned to the target page will be higher. Although primarily used for search engines, this algorithm can be applied in other contexts as well. In our specific case, we will focus on utilizing data from the Amazon US Customer Reviews dataset to classify products.

In Section 2, I will provide an overview of the algorithm's history and its mathematical derivation. Section 3 will explain how the algorithm has been applied specifically to Amazon reviews, highlighting how the solution obtained scales with the size of the data. Section 4 will introduce the dataset and outline the steps taken to calculate the PageRank for the products. Finally, in Section 5, we will present conclusions related to this project.

2 PageRank

The PageRank technique is a powerful algorithm employed by Google to organize web pages according to their relevance. Prior to the introduction of PageRank, search engine results were not very efficient, as they primarily relied on the search terms used. Page creators often resorted to keyword stuffing, where they would excessively use keywords within their pages to attract users. To address this issue, Google developed the PageRank algorithm, which simulates the behavior of random surfers.

The algorithm starts by imagining random surfers who begin their browsing journey from random web pages. These surfers then randomly choose one of the outgoing links on the page they started from and proceed to the linked page. By repeating this process, a state of convergence is reached, and pages that attract the most surfers are considered the most important or relevant.

The essence of PageRank lies in the idea that a page's importance is not solely determined by the number of incoming links but also by the quality and relevance of those incoming links. Pages with a higher PageRank score are seen as more influential and trustworthy, as they have a greater number of reputable pages linking to them. This approach helps mitigate the impact of keyword manipulation and promotes the ranking of genuinely valuable content.

In the next section, I will delve into the mathematical aspects of the PageRank algorithm and explain how it computes the relevance of web pages.

2.1 The algorithm

The PageRank algorithm assigns a numerical value to each web page, indicating its importance. The higher the assigned value, the more significant the page is considered. The algorithm takes into account the links between pages: a page can be reached from other

pages if they have links pointing to it, and vice versa.

Mathematically, the structure of the web can be represented by a transition matrix $\mathbf{M}$ of size $n \times n$, where n is the total number of pages on the internet. The matrix $\mathbf{M}$ contains the probabilities of transitioning from one page (node) to another. As mentioned in Section 1, a random surfer can start from any page, so at the beginning of the process, the probability of being on any of the n pages is equal to $1/n$. This initial probability distribution is represented by a row vector $\boldsymbol{v}_0$ of size $1 \times n$, where all entries are equal to $1/n$.

During each step of the algorithm, the random surfer moves from the current page to a new page by following one of the outgoing links from the current page. This transition is determined probabilistically based on the entries in the transition matrix $\mathbf{M}$. After the first step, the probability vector becomes equal to $\mathbf{M}\boldsymbol{v}_0$. At the second step, it becomes $\mathbf{M}(\mathbf{M}\boldsymbol{v}_0) = \mathbf{M}^2\boldsymbol{v}_0$. Continuing this process, at the i-th step, the probability vector will be $\mathbf{M}^i\boldsymbol{v}_0$.

The iteration continues until a state of convergence is reached, where the probability vector stabilizes. This means that the probabilities of being on different pages stop changing significantly. The final probability vector represents the PageRank scores of the pages, indicating their relative importance.

In the next section, I will explain how the PageRank algorithm has been applied specifically in the context of Amazon reviews and how the solution obtained scales with the size of the data.

3 PageRank applied to the Amazon Reviews dataset

The PageRank algorithm can be effectively applied to recommendation systems, and in our case, we will focus on its application to sort products on the Amazon platform. In this context, the products themselves act as nodes, and a link is created between two products if they have been reviewed by at least the same customer.

One might question why simply considering the number of real reviews for individual products is not sufficient for sorting them. If products were ranked solely based on the number of reviews, it could lead to the issue of fake reviews being used to artificially boost the visibility of certain products. However, by applying the PageRank algorithm, this problem can be mitigated. To artificially increase the rank of a specific product, fake reviews would need to be generated for numerous other products as well. Considering the vast user base of Amazon, the weight assigned to a product depends on the reviews provided by other genuine users, making it more resilient to manipulation.

3.1 Big Data: management techniques and tools

The PageRank algorithm is commonly applied to large datasets, and in our case, the dataset has a size of 54.41 GB. However, for the purpose of analysis, as explained later, a smaller subset was chosen. Nevertheless, the techniques employed in this project can be applied to large datasets as well.

When dealing with large data processing, one of the available solutions is Apache Spark, an open-source big data processing engine. Apache Spark enables distributed processing across multiple clusters, allowing for parallel execution. A SparkContext is created, which works with the cluster manager and distributes the workload among the nodes. One of the advantages of working with distributed nodes is that if a particular machine fails, the work can still be completed by the other nodes where the data is replicated.

The data is parallelized into an RDD (Resilient Distributed Dataset), which is a typical structure used in Spark. RDDs support various transformation operations, allowing for efficient data manipulation and analysis. For more details on RDDs and their usage, it is recommended to refer to the RDD Programming Guide.

4 Link analysis: implementation and results

4.1 Dataset description

The original dataset comprises 37 CSV files, with each file corresponding to a specific product category. Due to time constraints, the focus of this project was narrowed down to a single product category, namely "Electronics." However, the code can be modified to incorporate additional product categories if desired.

Each file in the dataset contains 15 columns, but for the purpose of analysis, only the following columns were considered:

- *customer_id*: a unique identifier for each customer;
- *review_id*: a unique identifier for each review;
- *product_id*: a unique identifier for each product;
- *product_title*: the name of the product.

In order to reduce processing times and obtain easily interpretable PageRank results, only 100,000 reviews were considered for analysis. It is worth noting that this parameter can be adjusted to include a different number of reviews if needed.

4.2 Data exploration

The exploratory analysis was conducted to identify products with the highest number of reviews, allowing for a comparison with products that have the highest PageRank. While this analysis is not necessary for calculating PageRank, it provides valuable insights. Due

to the limitations of working with large datasets, the analysis was performed on a Pandas dataframe, which is not suitable for processing large datasets.

After ensuring that there were no null values in the dataframe, the products were sorted based on the number of reviews. Initially, the product title variable was considered for product identification. However, it was discovered that multiple product IDs could be associated with a single product title. Therefore, the product ID was chosen as the identifier.

The top four products based on the number of reviews are as follows:

1. Product ID: B00F5NE2KG, 513 reviews;
2. Product ID: B003EM8008, 467 reviews;
3. Product ID: B003L1ZYYM, 455 reviews;
4. Product ID: B000WYVBR0, 289 reviews.

Figure 1 displays additional products sorted by the number of reviews.

4.3 PageRank computation

Once the exploratory analysis was completed, the focus shifted to calculating PageRank, which involves several steps as listed below:

1. Creation of nodes and arcs;
2. Creation of the transition matrix and the initial random surfer probability vector;
3. PageRank calculation.

Now we will focus on one step at a time.

4.3.1 Nodes and arcs

To begin, the Pandas dataframe was converted into a Spark dataframe. Although it's possible to directly load the dataframe with Spark, for the sake of flexibility in working with small dataframes, I chose to perform the conversion separately.

The nodes in this context represent the products that have been reviewed by at least one customer. To identify the customers who have reviewed at least two products, the following code snippet was used:

```
customer_review = sdf.rdd.map(lambda x: (x[0], x[2])).groupByKey()\
                         .map(lambda x : (x[0],list(x[1])))\
                         .filter(lambda x: len(x[1])>=2)
```

The Spark dataframe was transformed into an RDD, and the *customer_id* and *product_id* variables were selected. Key-value pairs were then created, where the customer ID served as the key and the values were lists of products reviewed by each customer. Finally, a filter

operation was applied to consider only those key-value pairs whose value (i.e., the list of products) had a length greater than or equal to 2.

The itertools module was utilized to generate the arcs. A function named combination was defined to consider the second element of the RDD pipeline, namely the products reviewed by customers. The permutations function from itertools was used to create tuples containing pairs of products in all possible orderings without repetitions (e.g., if a customer reviewed products A and B, tuples (A, B) and (B, A) would be created, while tuples (A, A) and (B, B) would be excluded).

The following code snippet demonstrates the process:

```
import itertools
def combination(row):
    l = row[1]
    res1 = [(v[0], v[1]) for v in itertools.permutations(l, 2)]
    return (res1)

edges = customer_review.map(lambda x: combination(x)).flatMap(lambda l
    ↪ :l)
```

The number of nodes in this case is 11,870, while the number of edges is 56,144.

4.3.2 Connection matrix and PageRank

To calculate the transition matrix, we need to determine the number of outgoing arcs for each node. This can be achieved by applying the countByKey function on the edges RDD, which results in a dictionary where the individual nodes are keys and the count of associated products is the corresponding value.

Next, the dictionary is converted into a list sorted by the number of links:

```
id2degree = edges.countByKey()
id2degree_list = sorted(id2degree.items(), key=lambda x: x[1], reverse
    ↪ =True)
```

The transition matrix is represented by tuples of three elements: the nodes, the products connected to the nodes, and the probability of reaching the products starting from the node, which is calculated as $1/id2degree$. As mentioned earlier, we will need to multiply the matrix by the probability vector, so we consider its transpose.

For the calculation of the probability vector, we create the variable p, which represents the initial probability of ending up in one of the nodes (given by $1/nodes$), and then create a list of products. The vector of initial probabilities is represented by a dictionary with the

products as keys and the probability as the value.

Multiplying the matrix and the vector will yield a new vector, which is then multiplied by the matrix again, and so on. After a certain number of iterations, the vector will converge, and the final result will provide the PageRank values associated with each product.

In the code provided below, we consider the product codes and the corresponding probabilities of ending up in those nodes given the other products. We multiply these probabilities by the initial probability vector and obtain a data list from the product-probability pairs. This list is then used to update the initially empty dictionary. The process is iterated 100 times.

```
P = edges.map(lambda x:(x[0],x[1],1/id2degree[x[0]]))
PT = P.map(lambda x: (x[1],x[0],x[2]))
p = 1/(nodes)

products = customer_review.flatMap(lambda x: x[1]).collect()
pdiz = {}
for prod in products:
    pdiz[prod] = p

for i in range(100):
    new_p = PT.map(lambda x:(x[0],(x[2]*pdiz[x[1]])))\
              .reduceByKey(lambda x,y: x+y)\
              .collect()
    for idx,prb in new_p:
        pdiz[idx] = prb
```

4.4 An alternative approach

In this section, we explore an alternative approach to calculate the PageRank values.

To begin, we create a dictionary that indexes the products. Each product code from the products list is associated with its corresponding index number. We also create a list of the same length as the products list, where all elements are initialized to a value denoted as p (defined as 1 divided by the number of nodes). This p represents the initial probability of landing on any node.

The matrices P and PT are defined similarly to the previous section, but instead of using product codes, we use the indexes obtained from the dictionary. Additionally, we introduce the l2distance function, which calculates the Euclidean distance between two elements. This function is necessary for calculating the final PageRank.

The PageRank is calculated as follows:

- We define a tolerance variable with a value of 10e-7, which represents the threshold for determining convergence of PageRank scores;
- We set a max_iterations variable with a value of 1000, representing the maximum number of iterations allowed for calculating PageRank scores;
- The page_rank vector is initialized with values equal to 1 divided by the total number of nodes. This vector represents the initial PageRank scores for each node;
- The old_page_rank vector is initialized with values all equal to 1. This vector will store the PageRank scores from the previous iteration;
- An iteration variable is initialized with a value of 0 to keep track of the number of iterations so far.

Next, we enter a while loop that continues as long as the Euclidean distance between the old PageRank scores and the new PageRank scores is greater than or equal to the tolerance, and the number of iterations is less than the maximum allowed iterations.

Within the while loop:

- The old_page_rank variable is updated with the current PageRank scores;
- The elements of the PT list are mapped to pairs (t[0], t[2] * page_rank[t[1]]). Here, the current PageRank score is multiplied by the value of t[2], which represents the initial probability. The values are then reduced by key, summing the corresponding values for each key. This aggregates the PageRank scores for each node;
- The results are sorted by key, and the values are collected and assigned to the page_rank_values variable as a list of tuples;
- The page_rank variable is updated with the PageRank score values from the page_rank_values list;
- The current PageRank vector is printed;
- The iteration variable is incremented by 1.

The while loop continues until the convergence condition or the iteration limit is reached. In this case, convergence is achieved after 25 iterations.

Here below the code is shown:

```
tolerance = 10e-7
max_iterations = 1000
page_rank = [p for x in range(len(products))]
old_page_rank = [1 for x in range(len(products))]

iteration = 0
```

```
while l2distance(old_page_rank, page_rank) >= tolerance and \
      iteration < max_iterations:
    old_page_rank = page_rank

    page_rank_values = (PT
                        .map(lambda t: (t[0], t[2]*page_rank[t[1]]))
                        .reduceByKey(lambda a, b: a+b)
                        .sortByKey()
                        .collect()
                       )

    page_rank = np.array([c for (i, c) in page_rank_values])

    print(page_rank)

    iteration += 1
```

---

5 Final results and conclusion

In Figure 2, we observe the final PageRank results. It is evident that the PageRank values are relatively low, which is expected. As the number of nodes and connections increases, the final probability values tend to approach zero. It is important to note that for the purpose of this analysis, the dataset has been reduced to ensure interpretability of the results.

One notable aspect to consider is that the products with the highest PageRank values do not necessarily correspond to the most reviewed products. For instance, even though product B003L1ZYYM ranks third in terms of reviews, it has the highest PageRank value. Conversely, the most reviewed product (B00F5NE2KG) does not appear in the top 20 PageRank products. This discrepancy arises because the algorithm takes into account the number of connections linking a product, rather than the sheer number of reviews. Therefore, this approach proves valuable in detecting products associated with fake reviews.

This analysis represents just an initial step towards developing a comprehensive algorithm that classifies products based on their importance, which is determined by the reliability of the reviewer. To further enhance query results for consumer needs, it may be beneficial to focus on products with a high number of reviews and apply additional filters to display results tailored to user interests.

List of figures

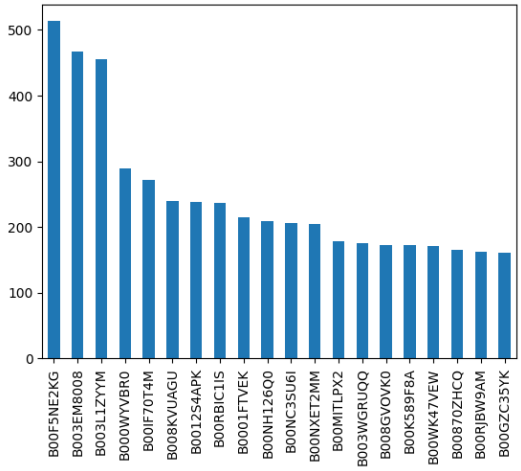

Figure 1: Top 20 most reviewed Products

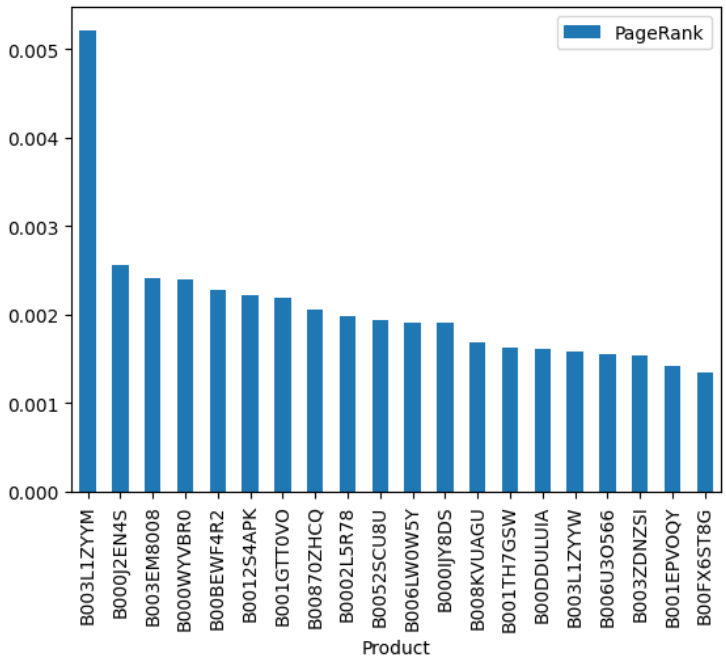

Figure 2: Top 20 PageRank Products