Assignment 3. Data matching

COMPSCI 767

Assignment 3. Data matching

COMPSCI 767

Juan David Roa Valencia

Data Loading and Preliminary Analysis

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

table_a = pd.read_csv('tableA/tableA.csv')
table_b = pd.read_csv('tableB/tableB.csv')

table_a.columns = ['ltable_' + str(col) for col in table_a.columns]
table_b.columns = ['rtable_' + str(col) for col in table_b.columns]

G = nx.Graph()

for index, row in table_a.iterrows():
    G.add_node(row['ltable_ID'], label=row['ltable_title'], dataset='A')

for index, row in table_b.iterrows():
    G.add_node(row['rtable_ID'], label=row['rtable_title'], dataset='B')

for index_a, row_a in table_a.iterrows():
    if isinstance(row_a['ltable_authors'], str):
        authors_a = row_a['ltable_authors'].split(', ')
        for index_b, row_b in table_b.iterrows():
            if isinstance(row_b['rtable_authors'], str):
                authors_b = row_b['rtable_authors'].split(', ')
                common_authors = set(authors_a).intersection(set(authors_b))
                if common_authors:
                    G.add_edge(row_a['ltable_ID'], row_b['rtable_ID'])

pr = nx.pagerank(G, alpha=0.9)

table_c = pd.DataFrame({'node': list(pr.keys()), 'PageRank': list(pr.values())})

tableC = pd.merge(table_c, table_a, how='left', left_on='node', right_on='ltable_ID')
tableC = pd.merge(tableC, table_b, how='left', left_on='node', right_on='rtable_ID')

tableC.fillna('Unknown', inplace=True)

tableC.to_csv('tableC.csv', index=False)

print(tableC.head())
```

Plotting graph representation

```
pagerank_threshold = 0.001

H = G.subgraph([node for node, pagerank in pr.items() if pagerank > pagerank_threshold])

pos = nx.spring_layout(H, seed=42)
plt.figure(figsize=(12, 12))
nx.draw_networkx(H, pos, with_labels=True, node_size=1000)
plt.show()
```

Time complexity estimation approximation based on used algorithms

Adding Nodes: $O(n + m)$ where n is the number of elements in `tableA` and m is the number of elements in `tableB`.

Adding Edges: $O(n * m * k * l)$ where n and m are the number of elements in `tableA` and `tableB`, respectively, and k and l are the average lengths of the author lists in `tableA` and `tableB`, respectively.

PageRank Calculation: $O(I * a)$ where I is the number of iterations and a is the number of edges in the graph.

Therefore, the overall complexity would be $O(n + m + n * m * k * l + I * a)$. But if $n * m * k * l$ is the highest term, it could be simplified to $O(n * m * k * l)$.

Plotting time complexity

```
import matplotlib.pyplot as plt
import numpy as np

k = 5
l = 5
```

```
I = 20

n = 1000
m = 1000

a = n*m*k*l

node_complexity = n + m
edge_complexity = n * m * k * l
pagerank_complexity = I * a

total_complexity = node_complexity + edge_complexity + pagerank_complexity

fig, ax = plt.subplots()

x = np.array([0, 1000])

y1 = np.full_like(x, node_complexity)
y2 = np.full_like(x, edge_complexity)
y3 = np.full_like(x, pagerank_complexity)
yt = np.full_like(x, total_complexity)

ax.plot(x, y1, label='Adding Nodes')
ax.plot(x, y2, label='Adding Edges')
ax.plot(x, y3, label='PageRank Calculation')
ax.plot(x, yt, label='Total', linestyle='--')

ax.set_xlabel('Operations Count')
ax.set_ylabel('Time Complexity (Operations)')
ax.set_title('Time Complexity Estimation')
ax.legend()

plt.show()
```

[TimeComplexity]({{ "/assets/img/complexity.png" | relative_url }})

We can conclude that the most expensive step and algorithm is the Page Rank.

Algorithms for Data Matching

Levenshtein Distance Based on the National Institute of Standards and Technology, the Levenshtein Distance is defined as the smallest number of insertions, deletions, and substitutions required to change one string or tree into another. A $\Theta(m \times n)$ algorithm to compute the distance between strings, where m and n are the lengths of the strings.

In this case, where we have a set of authors, it may not be as effective as it would not capture the connections between different papers based on common authors.

$$D[i,j] = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+1 & \text{if } a[i] \neq b[j] \\ D[i-1,j-1] & \text{if } a[i] = b[j] \end{cases} & \text{otherwise} \end{cases}$$

Cosine Similarity This measure calculates the cosine of the angle between two non-zero vectors, and it is widely used in document similarity and text mining. However, its usage in our scenario would have been more complex. Cosine similarity typically works on numerical vectors, so we must first convert our author data into a suitable numerical format.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

PageRank Based on Python Pandas definition, it computes a ranking of the nodes in a graph based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages. The PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number $L(v)$ of links from page v.

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

- $PR(u)$ is the PageRank of u
- Bu is the set of backlinks of u
- $L(v)$ is the number of links from page v
- N is the total number of pages
- d is the damping factor, usually set to 0.85

The PageRank algorithm is iterative and converges, so the time complexity is not constant. However, it is generally considered efficient for large-scale problems and is usually implemented on distributed systems for massive graphs.

{: .box-note} **Selected Algorithm:** The PageRank algorithm was preferred for its inherent suitability to the problem. The task involved identifying highly cited or highly connected papers based on common authors, which is essentially what PageRank was designed to do - identify highly connected nodes in a network.

{: .box-note} Moreover, it assigns a higher rank to nodes (papers) with high-quality connections (common authors), which was highly beneficial for our task. PageRank gives a single, quantitative measure for each paper that helped in matching.

Problems

The main problem faced was related to data cleaning and preparation. The data needed to be in the correct format to build the graph and calculate the PageRank. In particular, there were issues with non-string data types while attempting to split the authors. Other challenges faced while calculating PageRank based on the title, as it was leading to constant PageRank scores. This was because the titles of the papers were unique and hence, did not create any connections (edges) between the papers in the graph.

Report of `tableA`, `tableB` and `tableC` sizes

The size of `tableA` and `tableB` are both 1000 each, so the total number of tuple pairs in the Cartesian product of A and B is $1000 * 1000 = 1000000$. The total number of tuple pairs in `tableC` will be the number of edges in the graph, which depends on the number of common authors between different papers.

Additional cleaning

The author fields in both tables had to be in a string format that could be split into individual authors. This involved handling non-string data types and missing values.