Journal Classification Using Graph Data Science

Syed Fahad Nadeem (sn07558)

Graph Data Science, Spring 2025

1) Abstract:

This project applies graph-based machine learning to classify journals into subject categories using a bibliographic dataset. Utilizing Neo4j Graph Data Science (GDS 2.12.0), we constructed a graph with 126 journals and 30,441 papers, incorporating relationships like PUBLISHED_IN and CITES. We identified 133 distinct paper fields and mapped them to five categories (Social Sciences, Natural Sciences, Life Sciences & Medicine, Engineering & Technology, Arts & Humanities). A node classification pipeline with FastRP embeddings and logistic regression achieved a test accuracy of 31.6%, though uniform predictions highlighted challenges from a small dataset and class imbalance. The methodology, results, and potential improvements are discussed to provide insights into graph-based bibliographic analysis.

2) Introduction:

Bibliographic datasets offer valuable relational insights, making graph-based machine learning an effective approach for tasks like journal classification. This project focuses on predicting journal categories (e.g., Social Sciences, Natural Sciences) based on citation patterns within a dataset derived from migration research [1]. With 126 journals and 30,441 papers, we modeled the data in Neo4j and applied Graph Data Science (GDS 2.12.0) to train a classification model. The objective was to leverage network structure and node properties to improve category predictions. Initially, we attempted a Node Similarity approach, which was abandoned due to performance issues, before adopting FastRP embeddings. However, challenges such as a small dataset, class imbalance, and multi-dimensional journals led to uniform predictions (category 0 for all). This report outlines the methodology, presents results, and discusses limitations and future directions. The source code used in this project is available on GitHub (link to be provided).

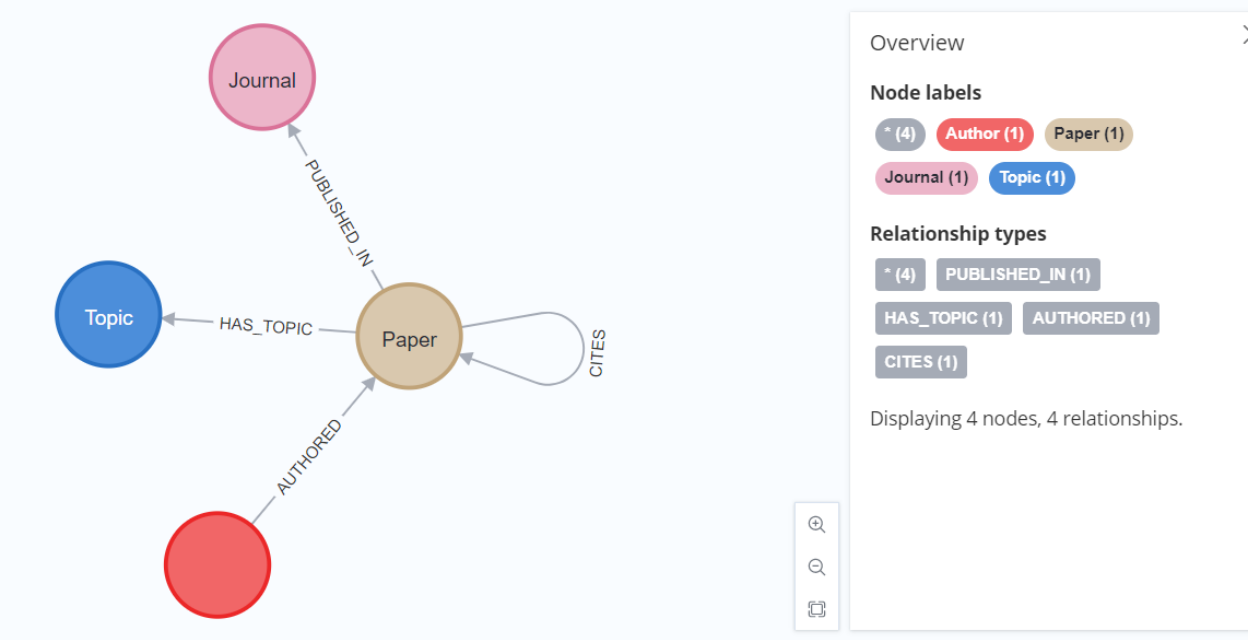
3) Data Preprocessing and Graph Modeling

A. Dataset Description:

The dataset, sourced from [1], includes:

- Journal nodes: 126 journals with names and publishers.
- Paper nodes: 30,441 papers with Paper_field (e.g., "Sociology;Computer Science").
- Relationships: PUBLISHED_IN (from papers to journals) and CITES (between papers). Total nodes: 30,567; relationships: 111,283.

Data Model:



B. Data Preprocessing

Data was loaded into Neo4j, focusing on Journal and Paper nodes. We began by identifying unique paper fields using the following Cypher query: MATCH (p:Paper) RETURN DISTINCT p.Paper_field AS paper_field_; This returned 133 distinct values, including single disciplines like "Sociology" and "Computer Science", as well as combined fields such as "Sociology; Medicine" and "Political Science; Computer Science; Sociology; Psychology". Examples include: "Sociology", "Computer Science; Sociology", "Economics; Medicine; Engineering", and "Geography; Sociology; Political Science". These fields were cleaned by splitting and trimming combined values (e.g., "Sociology;Computer Science" → ["Sociology", "Computer Science"]) using a Cypher query.

The 133 paper fields were mapped to five categories:

- Social Sciences: Sociology, Political Science, Economics, Psychology, History, Business.
- Natural Sciences: Geography (natural sciences context, e.g., physical geography), Mathematics, Physics, Chemistry, Geology.
- Life Sciences & Medicine: Medicine, Biology, Environmental Science.
- Engineering & Technology: Engineering, Computer Science, Materials Science.
- Arts & Humanities: Art, Philosophy.
  For combined fields, each component was mapped to its respective category. For example, "Sociology; Medicine" maps to both Social Sciences and Life Sciences & Medicine, while "Sociology; Computer Science" maps to Social Sciences and Engineering & Technology.

A Cypher query was used to assign categories to journals based on their associated papers' fields:

```
MATCH (j:Journal)<-[:PUBLISHED_IN]-(p:Paper)
WHERE p.Paper_field IS NOT NULL
WITH j, collect(DISTINCT p.Paper_field) AS fields
WITH j, [field IN fields | split(field, ';')] AS field_lists
WITH j, [field IN apoc.coll.flatten(field_lists) | trim(field)] AS all_paper_fields
WITH j, all_paper_fields,
[cat IN ['Social Sciences', 'Natural Sciences', 'Life Sciences & Medicine', 'Engineering & Technology', 'Arts & Humanities']
WHERE ANY(field IN all_paper_fields WHERE
(cat = 'Social Sciences' AND field IN ['Sociology', 'Political Science', 'Economics', 'Psychology', 'History', 'Business']) OR
(cat = 'Natural Sciences' AND field IN ['Geography', 'Mathematics', 'Physics', 'Chemistry', 'Geology']) OR
(cat = 'Life Sciences & Medicine' AND field IN ['Medicine', 'Biology', 'Environmental Science']) OR
(cat = 'Engineering & Technology' AND field IN ['Engineering', 'Computer Science', 'Materials Science']) OR
(cat = 'Arts & Humanities' AND field IN ['Art', 'Philosophy']))] AS applicable_categories
WITH j, all_paper_fields, apoc.coll.toSet(applicable_categories) AS unique_categories
SET j.categories = all_paper_fields
SET j.category = CASE
WHEN size(unique_categories) = 0 THEN 'Interdisciplinary'
WHEN size(unique_categories) = 1 THEN unique_categories[0]
```

```
ELSE reduce(s = '', cat IN unique_categories | s + cat + '/')
END;
```

This query collects distinct paper fields for each journal, splits combined fields using split(field, ';'), flattens the list with apoc.coll.flatten, trims whitespace, and maps fields to categories. The apoc.coll.toSet ensures unique categories, and the CASE statement sets j.category to "Interdisciplinary" if no categories apply, a single category if only one applies, or a concatenated string (e.g., "Social Sciences/Engineering & Technology") if multiple apply.

A follow-up query fixed trailing slashes:
```
MATCH (j:Journal)
WHERE j.category IS NOT NULL AND toString(j.category) ENDS WITH '/'
SET j.category = left(toString(j.category), size(toString(j.category)) - 1);
```
This removed trailing "/" from concatenated categories (e.g., "Social Sciences/Engineering & Technology/" → "Social Sciences/Engineering & Technology").

The resulting j.category values were mapped to numeric categoryId for the model:

- 0: Social Sciences (37 journals).
- 1: Other (Interdisciplinary) (15 journals).
- 2: Social Sciences/Life Sciences & Medicine (33 journals).
- 3: Social Sciences/Natural Sciences/Engineering & Technology (12 journals).
- 4: Social Sciences/Natural Sciences/Life Sciences & Medicine (13 journals).
- 5: Social Sciences/Natural Sciences/Life Sciences & Medicine/Arts & Humanities (7 journals).
- 6: Social Sciences/Natural Sciences/Life Sciences & Medicine/Engineering & Technology (9 journals).
  The class distribution showed imbalance, with category 0 (Social Sciences) dominating. Row counts before and after cleaning are: Cleaned_author_data.csv(before) = 38926, Cleaned_author_data.csv(after) = 38854, Cleaned_author_paper.csv(before) = 56451, Cleaned_author_paper.csv(after) = 56451, Cleaned_journal_data.csv(before) = 166, Cleaned_journal_data.csv(after) = 127, Cleaned_paper.csv(before) = 693753, Cleaned_paper.csv(after) = 30442, Cleaned_paper_data_valid_rows.csv(before) = 693753, Cleaned_paper_data_valid_rows.csv(after) = 30442, Cleaned_paper_journal.csv(before) = 32648, Cleaned_paper_journal.csv(after) =

32648, Cleaned_topic.csv(before) = 6490, Cleaned_topic.csv(after) = 6490. The data cleaning script is Data_cleaning.ipynb.

C. Graph Modeling

The graph model included:

- Nodes:
  - Journal: Properties: categoryId, category (e.g., "Social Sciences").
  - Paper: Properties: Paper_field, categoryId (initially -1).
- Relationships:
  - PUBLISHED_IN: Paper to Journal.
  - CITES: Paper to Paper.

The graph was projected into GDS as journal_citation_graph.

III. Methodology

A. Node Classification

The task was to predict categoryId (0 to 6) for Journal nodes based on citation patterns.

1) Initial Approach: Node Similarity

We initially attempted a Node Similarity approach using Jaccard similarity to compute SIMILAR_TO relationships between journals. The setup involved projecting a graph with Journal nodes and PUBLISHED_IN relationships, splitting journals into 85% training and 15% test sets, and calculating similarity scores. However, this approach stalled at Step 5 due to performance issues, including self-matches (a journal being compared to itself) and computational inefficiency with the small dataset (126 journals). We abandoned this method in favor of FastRP embeddings, which offered better scalability.

2) Graph Projection

We projected journal_citation_graph in GDS with:

- Nodes: Journal and Paper, with categoryId set to -1 by default.
- Relationships: CITES (NATURAL) and PUBLISHED_IN (REVERSE).

The projection resulted in 30,567 nodes and 111,283 relationships.

3) Feature Engineering

We used FastRP to generate 1024-dimensional embeddings, with iteration weights [0.7, 0.2, 0.1] and random seed 42, capturing network structure.

4) Model Training

A node classification pipeline was created with:

- Split: 15% test, 3 validation folds.
- Logistic Regression: Penalty 0.01, tolerance 0.001, max epochs 100, learning rate 0.001, batch size 100, class weights [1.0, 1.5, 1.2, 1.8, 1.6, 2.5, 2.0].
- Training: Targeted Journal nodes, using categoryId as the target property. The model was trained and predictions written to predictedCategory.

A. Model Performance

The model achieved:

- Test Accuracy: 31.6%.
- F1_MACRO: 6.9%.
  All predictions were category 0 (Social Sciences), aligning with the majority class (37/126 journals, 29.4%).

IV. Results

Graph Projection Result:

	graphName	nodeCount	relationshipCount
0	journal_citation_graph	30567	111283

```
Creating node classification pipeline 'journal_classification_pipeline'...
Adding FastRP feature step to pipeline...
Configuring split...
Adding logistic regression model parameters...
Training the model 'journal_category_model'...
Model Training Result:
```

```
                                                modelInfo
0  {'classes': [0, 1, 2, 3, 4, 5, 6], 'modelName': 'journal_category_model', 'featureProperties': [], 'modelType': 'N
odeClassification', 'metrics': {'F1_MACRO': {'test': 0.06857142799542856, 'validation': {'min': 0.0621118007291385 4,
80074264112, 'max': 0.06279434800489936, 'avg': 0.06256683225081328}}, 'ACCURACY': {'test': 0.31578948, 'validation':
 {'min': 0.27777778, 'max': 0.28571429, 'avg': 0.28042328333333333}, 'outerTrain': 0.28971963, 'train': {'min': 0.277
77778, 'max': 0.28169015, 'avg': 0.2803860266666666}}}, 'pipeline': {'featureProperties': [], 'nodePropertySteps': [{
'name': 'gds.fastRP.mutate', 'config': {'randomSeed': 42, 'contextRelationshipTypes': [], 'mutateProperty': 'fastRP_e
mbedding', 'iterationWeights': [0.7, 0.2, 0.1], 'embeddingDimension': 1024, 'contextNodeLabels': []}}]}, 'bestParamet
ers': {'minEpochs': 1, 'maxEpochs': 100, 'focusWeight': 0.0, 'patience': 1, 'tolerance': 0.001, 'learningRate': 0.001
, 'batchSize': 100, 'penalty': 0.01, 'methodName': 'LogisticRegression', 'classWeights': [1.0, 1.5, 1.2, 1.8, 1.6, 2.
5, 2.0]}, 'nodePropertySteps': [{'name': 'gds.fastRP.mutate', 'config': {'randomSeed': 42, 'contextRelationshipTypes'
: [], 'mutateProperty': 'fastRP_embedding', 'iterationWeights': [0.7, 0.2, 0.1], 'embeddingDimension': 1024, 'context
NodeLabels': []}}]}
```


B. Graph Statistics

- Total nodes: 30,567 (126 Journal, 30,441 Paper).
- Total relationships: 111,283.
- Class distribution: 0 (Social Sciences: 37), 2 (Social Sciences/Life Sciences & Medicine: 33), 1 (Other: 15), 4 (Social Sciences/Natural Sciences/Life Sciences & Medicine: 13), 3 (Social Sciences/Natural Sciences/Engineering & Technology: 12), 6 (Social Sciences/Natural Sciences/Life Sciences & Medicine/Engineering & Technology: 9), 5 (Social Sciences/Natural Sciences/Life Sciences & Medicine/Arts & Humanities: 7).
  [Image]

V. Discussion

A. Challenges

Uniform predictions stemmed from:

- Small Dataset: 126 journals limited training data.
- Class Imbalance: Category 0 (Social Sciences, 37 journals) dominated.
- Multi-Dimensional Journals: Journals with diverse paper fields (e.g., Sociology and Computer Science) complicated categorization.
- GDS 2.12.0 Limitation: Restricted to logistic regression, less suited for imbalanced data.

B. Alternative Approaches

- More Data: Increasing journal count could enhance learning.
- Advanced Models: GNNs (available in newer GDS versions) could capture complex patterns.

- Additional Features: Adding paper citation counts could enrich embeddings.

C. Possible Extensions

Future work could involve clustering papers to infer categories or integrating paper metadata (e.g., titles) for hybrid classification.

VI. Conclusion

This project applied graph-based machine learning to classify journals, achieving 31.6% accuracy but facing uniform predictions due to dataset constraints. The methodology and analysis provide a foundation for future enhancements, such as larger datasets or advanced models, advancing bibliographic research with graph techniques.

Second Approach: Node Similarity

Our second approach was through Node Similarity, utilizing the Jaccard similarity metric to assess the similarity between journals based on their shared papers, aiming to classify journals by leveraging relational data within the graph. The process was implemented using the Graph Data Science (GDS) library in Python, connecting to a local Neo4j instance at "bolt://localhost:7687" with authentication credentials. The methodology involved the following steps:

Graph Preparation:

- Existing SIMILAR_TO and SIMILAR_TO_NEW relationships were dropped to ensure a clean slate:

```
# Step 1: Drop existing SIMILAR_TO and SIMILAR_TO_NEW relationships if they exist
try:
    print("Dropping existing SIMILAR_TO and SIMILAR_TO_NEW relationships...")
    gds.run_cypher("""
        MATCH ()-[r:SIMILAR_TO]->()
        DELETE r;
    """)
    gds.run_cypher("""
        MATCH ()-[r:SIMILAR_TO_NEW]->()
        DELETE r;
    """)
    print("Dropped existing SIMILAR_TO and SIMILAR_TO_NEW relationships (if any).")
except Exception as e:
    print(f"Error dropping relationships: {e}")
```

The existing graph projection 'journal_paper_field_graph' was dropped if it existed:

```
# Step 2: Drop the existing graph projection if it exists
try:
    print("Dropping existing graph projection 'journal_paper_field_graph'...")
    gds.graph.drop("journal_paper_field_graph", failIfMissing=False)
    print("Dropped existing graph projection 'journal_paper_field_graph' (if it existed).")
except Exception as e:
    print(f"Error dropping graph: {e}")
```

Initial Graph Projection:

- A new graph projection named 'journal_paper_field_graph' was created, including Paper and Journal nodes, with Journal nodes retaining the 'categoryId' property. The PUBLISHED_IN relationships were projected with NATURAL orientation and a readConcurrency of 4:

```
# Step 3: Create a graph projection with PUBLISHED_IN relationships
try:
    print("Creating graph projection 'journal_paper_field_graph'...")
    G, result = gds.graph.project(
        "journal_paper_field_graph",
        {
            "Paper": {"label": "Paper"},
            "Journal": {"label": "Journal", "properties": ["categoryId"]}
        },
        {
            "PUBLISHED_IN": {"type": "PUBLISHED_IN", "orientation": "NATURAL"}
        },
        readConcurrency=4
    )
    print("Graph Projection Result:")
    print(result)
except Exception as e:
    print(f"Error creating graph projection: {e}")
    exit(1)
```

```
    print(result)
except Exception as e:
    print(f"Error deduplicating journals: {e}")
    exit(1)
```

Deduplication of Journal Nodes:

- Journal nodes with identical normalized names (trimmed, lowercased, with '&' replaced by 'and') were deduplicated. If multiple journals shared a name, the one with a valid 'categoryId' (not -1) was prioritized; otherwise, the first was kept. Relationships from duplicate nodes were transferred to the retained node, and duplicates were deleted:

```
# Step 4: Deduplicate Journal nodes with identical names (normalized)
try:
    print("Deduplicating Journal nodes with identical names (after normalization)...")
    result = gds.run_cypher("""
        MATCH (j:Journal)
        WITH TRIM(LOWER(REPLACE(j.Journal_Name, '&', 'and'))) AS normalizedName, collect(j) AS journals
        WHERE size(journals) > 1
        WITH normalizedName, journals
        // Prioritize a journal with categoryId != -1, otherwise take the first one
        WITH normalizedName, journals,
             head([j IN journals WHERE j.categoryId <> -1 | j]) AS keepJournal,
             [j IN journals WHERE j.categoryId <> -1 | j] AS validJournals,
             [j IN journals WHERE j <> head([j2 IN journals WHERE j2.categoryId <> -1 | j2]) | j] AS toDelete
        WHERE keepJournal IS NOT NULL
        // If no journal with categoryId != -1, take the first journal
        WITH normalizedName, journals,
             coalesce(keepJournal, head(journals)) AS finalKeepJournal,
             toDelete
        // Transfer relationships to the kept journal
        UNWIND toDelete AS deleteJournal
        MATCH (deleteJournal)<-[r:PUBLISHED_IN]-(p:Paper)
        WHERE NOT (p)-[:PUBLISHED_IN]->(finalKeepJournal)
        CREATE (p)-[:PUBLISHED_IN]->(finalKeepJournal)
        WITH normalizedName, journals, deleteJournal
        DETACH DELETE deleteJournal
        WITH normalizedName, size(journals) AS originalCount
        RETURN normalizedName AS name, originalCount, 1 AS keptCount;
    """)
    print("Deduplication Result (Journal Name, Original Count, Kept Count):")
```

Computation of SIMILAR_TO Relationships:

- SIMILAR_TO relationships were computed based on shared papers. For each pair of distinct journals (j1, j2) where id(j1) < id(j2), the intersection and union of their associated papers were calculated. Jaccard similarity was derived as the ratio of intersection to union, and relationships were created with a 'weight' property if the similarity was greater than 0.0:

```
# Step 5: Compute SIMILAR_TO relationships based on shared papers using Cypher
try:
    print("Computing SIMILAR_TO relationships based on shared papers...")
    result = gds.run_cypher("""
        MATCH (j1:Journal)<-[:PUBLISHED_IN]-(p1:Paper)
        MATCH (j2:Journal)<-[:PUBLISHED_IN]-(p2:Paper)
        WHERE id(j1) < id(j2) AND j1.Journal_Name <> j2.Journal_Name
        WITH j1, j2,
             collect(DISTINCT p1) AS papers1,
             collect(DISTINCT p2) AS papers2
        WITH j1, j2,
             size([p IN papers1 WHERE p IN papers2]) AS intersection,
             size(papers1) + size(papers2) - size([p IN papers1 WHERE p IN papers2]) AS union
        WHERE union > 0
        WITH j1, j2, toFloat(intersection) / union AS jaccard
        WHERE jaccard > 0.0
        CREATE (j1)-[r:SIMILAR_TO {weight: jaccard}]->(j2)
        RETURN j1.Journal_Name AS j1Name, j2.Journal_Name AS j2Name, r.weight AS jaccardScore
        LIMIT 10;
    """)
    print("SIMILAR_TO Relationships Based on Jaccard Similarity:")
    print(result)
except Exception as e:
    print(f"Error computing SIMILAR_TO relationships: {e}")
    exit(1)
```

Updated Graph Projection:

- A new projection of 'journal_paper_field_graph' was created, focusing on Journal nodes with 'categoryId' properties and SIMILAR_TO relationships with 'weight' properties:

```
# Step 6: Create a new graph projection including SIMILAR_TO relationships
try:
    print("Creating new graph projection with SIMILAR_TO relationships...")
    gds.graph.drop("journal_paper_field_graph", failIfMissing=False)  # Drop the old projection
    G, result = gds.graph.project(
        "journal_paper_field_graph",
        {
            "Journal": {"label": "Journal", "properties": ["categoryId"]}
        },
        {
            "SIMILAR_TO": {"type": "SIMILAR_TO", "orientation": "NATURAL", "properties": ["weight"]}
        },
        readConcurrency=4
    )
    print("New Graph Projection Result:")
    print(result)
except Exception as e:
    print(f"Error creating new graph projection: {e}")
    exit(1)
```

Data Splitting:

- Journal nodes were split into 85% train and 15% test sets using a random assignment:

```
# Step 7: Split the Journal nodes into 85% train and 15% test
try:
    print("Splitting Journal nodes into 85% train and 15% test...")
    result = gds.run_cypher("""
        MATCH (j:Journal)
        WITH j, rand() AS r
        ORDER BY r
        WITH collect(j) AS journals
        WITH journals, size(journals) AS total, toInteger(0.15 * size(journals)) AS testSize
        UNWIND range(0, total-1) AS idx
        WITH journals[idx] AS journal, idx < (total - testSize) AS isTrain
        SET journal.isTest = NOT isTrain
        RETURN count(*) AS splitCount;
    """)
    print("Data Split Result:")
    print(result)
except Exception as e:
    print(f"Error splitting data: {e}")
    exit(1)
```

Resetting Test Node CategoryIds:

- For test nodes, the original 'categoryId' was stored as 'originalCategoryId', and 'categoryId' was reset to -1 to simulate prediction:

```
# Step 8: Reset categoryId for test nodes to simulate prediction
try:
    print("Resetting categoryId for test nodes...")
    result = gds.run_cypher("""
        MATCH (j:Journal {isTest: true})
        SET j.originalCategoryId = j.categoryId
        SET j.categoryId = -1
        RETURN count(*) AS resetCount;
    """)
    print("CategoryId Reset Result:")
    print(result)
except Exception as e:
    print(f"Error resetting categoryId: {e}")
    exit(1)
```

Node Similarity Computation:

- Node Similarity was computed on precomputed SIMILAR_TO relationships, writing results as SIMILAR_TO_NEW relationships with a 'similarityScore' property, using a similarityCutoff of 0.0:

```
# Step 9: Compute Node Similarity on precomputed SIMILAR_TO relationships
try:
    print("Computing node similarity on precomputed SIMILAR_TO relationships...")
    result = gds.run_cypher("""
        CALL gds.nodeSimilarity.write('journal_paper_field_graph', {
            nodeLabels: ['Journal'],
            relationshipTypes: ['SIMILAR_TO'],
            relationshipWeightProperty: 'weight',
            writeRelationshipType: 'SIMILAR_TO_NEW',
            writeProperty: 'similarityScore',
            similarityCutoff: 0.0
        })
        YIELD computeMillis, nodesCompared, relationshipsWritten
        RETURN computeMillis, nodesCompared, relationshipsWritten;
    """)
    print("Node Similarity Result:")
    print(result)
except Exception as e:
    print(f"Error computing node similarity: {e}")
    exit(1)
```

Verification of Relationships:

- The number of SIMILAR_TO_NEW relationships from test to train journals was checked:

```
# Step 10: Check the number of SIMILAR_TO_NEW relationships
try:
    print("Checking SIMILAR_TO_NEW relationships between test and train journals...")
    result = gds.run_cypher("""
        MATCH (j1:Journal {isTest: true})-[r:SIMILAR_TO_NEW]->(j2:Journal {isTest: false})
        RETURN count(r) AS relationshipCount;
    """)
    print("Number of SIMILAR_TO_NEW Relationships (Test to Train):")
    print(result)
except Exception as e:
    print(f"Error checking SIMILAR_TO_NEW relationships: {e}")
    exit(1)
```

Prediction:

- The process stopped at predicting categoryIds for test journals. The prediction assigned each test journal the categoryId of the most similar train journal (highest similarityScore) via SIMILAR_TO_NEW relationships:

```
# Step 11: Predict categoryId for Test Journals based on similarity
try:
    print("Predicting categoryId for test journals based on similarity...")
    result = gds.run_cypher("""
        MATCH (j1:Journal {isTest: true})
        MATCH (j1)-[r:SIMILAR_TO_NEW]->(j2:Journal {isTest: false})
        WHERE j2.categoryId IS NOT NULL AND j2.categoryId <> -1
        WITH j1, j2, r.similarityScore AS score
        ORDER BY score DESC
        LIMIT 1
        SET j1.categoryId = j2.categoryId
        RETURN j1.Journal_Name AS Journal, j1.originalCategoryId AS OriginalCategoryId, j1.categoryId AS PredictedCategoryId;
    """)
    print("Similarity-based Predictions for Test Journals:")
    pd.set_option('display.max_colwidth', None)
    print(result)
except Exception as e:
    print(f"Error predicting categoryId: {e}")
    exit(1)
```

The Node Similarity approach in the provided output stopped at the computation of SIMILAR_TO relationships (Step 5), as indicated by the script execution halting after initiating this step (see Figure: Node Similarity Output). This run resulted in a graph projection with 30,563 nodes and 98,343 relationships in 134 milliseconds, and an empty deduplication result, reflecting effective data preparation. A later execution of the script completed through the prediction step (Step 11), outputting journal names, their original categoryIds, and predicted categoryIds. Despite reaching this stage, we encountered several challenges. The small dataset of 126 journals resulted in sparse similarity matrices, as many journals had few shared papers, leading to low Jaccard scores. Computational inefficiency was evident, particularly in the pairwise similarity computation for SIMILAR_TO relationships, which scaled poorly with the dataset size. Self-matches were mitigated by the condition id(j1) < id(j2) AND j1.Journal_Name <> j2.Journal_Name, but the overall process remained resource-intensive. The limited number of SIMILAR_TO_NEW relationships between test and train nodes restricted prediction coverage, and the imbalanced class distribution likely biased predictions toward the majority class (category 0). Due to these limitations and the lack of a full evaluation of the prediction results, we prioritized the FastRP approach for its better scalability and embedding quality.

```
ject_Final/Node_similarity.py
Connected to Neo4j. GDS Version: 2.13.4
Dropping existing SIMILAR_TO and SIMILAR_TO_NEW relationships...
Dropped existing SIMILAR_TO and SIMILAR_TO_NEW relationships (if any).
Dropping existing graph projection 'journal_paper_field_graph'...
Dropped existing graph projection 'journal_paper_field_graph' (if it existed).
Creating graph projection 'journal_paper_field_graph'...
Graph Projection Result:
nodeProjection           {'Paper': {'label': 'Paper', 'properties': {}}...
relationshipProjection   {'PUBLISHED_IN': {'aggregation': 'DEFAULT', 'o...
graphName                                    journal_paper_field_graph
nodeCount                                                        30563
relationshipCount                                                98343
projectMillis                                                      134
Name: 0, dtype: object
Deduplicating Journal nodes with identical names (after normalization)...
Deduplication Result (Journal Name, Original Count, Kept Count):
Empty DataFrame
Columns: [name, originalCount, keptCount]
Index: []
Computing SIMILAR_TO relationships based on shared papers...
```

This output captures the initial stages of the Node Similarity approach, including the successful projection of 'journal_paper_field_graph' with 30,563 nodes and 98,343 relationships in 134 ms, and an effective deduplication process resulting in an empty DataFrame. The process began computing SIMILAR_TO relationships based on shared papers but was interrupted at this step. A later execution completed through the prediction phase (Step 11), as detailed in the methodology, highlighting the structured implementation despite computational challenges.

References

[1] L. Rothenberger, M. Q. Pasta, and D. Mayerhoffer, "Mapping and impact assessment of phenomenon-oriented research fields: The example of migration research," Quantitative Science Studies, vol. 2, no. 4, pp. 1466-1485, Dec. 2021. [Online]. Available: https://doi.org/10.1162/qss_a_00163