Using Neo4j Graph Data Science in Python to Improve Machine Learning Models for Bibliographic Data

Sajal Fatima
Computer Engineering
Habib University
Karachi, Pakistan
sf08108@st.habib.edu.pk

Shaaf Farooque
Computer Engineering
Habib University
Karachi, Pakistan
sf08405@st.habib.edu.pk

***Abstract*—This project implements graph-based machine learning techniques for bibliographic network analysis using Neo4j and Python. We address two critical tasks: (1) node classification to predict author research domains through co-authorship networks, and (2) link prediction for citation recommendations using paper citation graphs. Our pipeline processes raw bibliographic metadata from CSV files to construct a Neo4j knowledge graph. For author classification, we extract graph features (PageRank, centrality measures, Jaccard similarity, Louvain communities) and train a Random Forest classifier, achieving 0.67 ROC-AUC with 75% precision - a 23% improvement over baseline. For citation recommendation, we engineer structural features (common neighbors, Adamic-Adar) and implement LightGBM with SMOTE, reaching 0.69 ROC-AUC with 90% precision - a 30% improvement from baseline. The results demonstrate that graph topology features significantly enhance prediction quality for both tasks, though class imbalance remains challenging for rare research domains and citations. Our reproducible workflow (available on GitHub) provides a template for bibliographic graph analysis with Neo4j GDS and scikit-learn integration.**

***Index Terms*—Graph Data Science, Neo4j, Machine Learning, Author Classification, Co-Authorship Network, Feature Engineering, PageRank, Closeness Centrality, Link Prediction, Citation recommendation**

I. Introduction

In recent years, graph-based machine learning has gained significant attention for analyzing complex relational data, particularly in bibliographic networks. Traditional machine learning models often struggle to capture the intricate relationships between scholarly entities like authors, papers, and journals. This project applies graph-based approaches to two fundamental tasks in academic knowledge discovery: (1) **node classification** to predict author research domains based on co-authorship networks, and (2) **link prediction** to recommend relevant citations through paper citation graphs.

Our work makes three key contributions:

- A comprehensive pipeline for constructing bibliographic knowledge graphs from heterogeneous scholarly data
- Demonstrated effectiveness of graph algorithms (PageRank, centrality measures, community detection) in enhancing both node classification and link prediction tasks
- Quantitative improvements over baseline methods, with our feature-enhanced models achieving 23% higher ROC-AUC for author classification and 30% better citation recommendation accuracy

The resulting framework enables more accurate modeling of academic relationships while providing interpretable features for scholarly network analysis. We implement our approach using Neo4j Graph Data Science with Python integration, ensuring reproducibility and extensibility for future research.

II. Methodology

A. Data Preprocessing

Data preprocessing is a critical step in ensuring that the dataset [1] is cleaned, consistent, and ready for graph construction and model training. This section outlines the preprocessing steps applied to the dataset, including `author.csv`, `paper.csv`, `journal.csv`, `paper_journal.csv`, and `topic.csv`.

For each of the CSV file, the following general preprocessing steps were applied:

1. Loading the Data: All files were loaded into pandas DataFrames using `pd.read_csv()` for initial inspection and processing.

2. Initial Exploration:

- The shape of each file was checked using `.shape` to understand the number of records.
- The first few rows were displayed using `.head()` to provide a quick view of the data.
- The data types of each column were reviewed to ensure that each feature was correctly formatted for further processing.

3. Handling Missing Values: Missing values were detected using `.isnull().sum()`. Columns with a significant number of missing values were addressed by either removing rows or imputing missing values. For instance, rows with missing Author URL were removed as they were critical for the analysis.

4. Duplicate Detection and Removal: The presence of duplicate rows was checked using `.duplicated().sum()`. Duplicate rows were removed to avoid redundancy.

5. Unique Value Counts: The number of unique values in key columns such as Author ID, Author Name, and Author URL was checked. This helped identify any unexpected or inconsistent data.

6. Exporting Cleaned Data: After preprocessing, each data file was saved as a new CSV file (`data_cleaned.csv`) to prepare it for further analysis.

7. Specific Cleaning Steps per file: The following specific cleaning steps were performed on each file:

- `authors.csv`: Missing Author URL values were removed, and duplicate authors were checked and eliminated. We ensured that Author IDs contained only valid numeric values.
- `paper.csv`: Missing citation counts were filled with zero, and other missing values were handled similarly. We ensured that Paper IDs and DOIs were valid.
- `journal.csv`: Missing Journal Publisher values were filled with "-", and duplicate journals were removed to maintain consistency.
- `paper_journal.csv`: We validated the relationships between papers and journals to ensure that all entries correctly linked papers to their respective journals.
- `topic.csv` and `paper_topic.csv`: We ensured that topics were properly linked to papers and that missing or invalid data points were handled appropriately.

B. Dataset and Graph Schema

The dataset used for this study [1] contains information on authors, papers, journals, and topics. It is structured into several CSV files:

- `author.csv`: Contains `Author_ID`, `Author_Name`, and other author metadata.
- `journal.csv`: Contains `Journal_Name` and `Publisher`.
- `paper.csv`: Contains `Paper_ID`, `Title`, `DOI`, `Year`, `Citations`, and other publication metadata.
- `topic.csv`: Contains `Topic_ID`, `Topic_Name`, and other metadata.
- `author_paper.csv`: Maps authors to papers via `Author_ID` and `Paper_ID`.
- `paper_journal.csv`: Links papers to journals via `Paper_ID` and `Journal_Name`.
- `paper_reference.csv`: Connects citing papers to cited papers via `Source_Paper_ID` and `Target_Paper_ID`.
- `paper_topic.csv`: Associates papers with topics via `Paper_ID` and `Topic_ID`.

We construct a knowledge graph to model scholarly entities and their relationships. The graph consists of the following components:

Nodes:

- **Author**: Represents a researcher or co-author.
- **Paper**: Represents a published research work.
- **Topic**: Represents a subject or field discussed in a paper.
- **Journal**: Represents the publication venue.

Relationships:

- An `Author` is connected to a `Paper` via a `CONTRIBUTED_TO` edge, indicating authorship. Multiple authors may contribute to the same paper (co-authorship).
- A `Paper` may cite another `Paper` through a `REFERS_TO` edge, forming a citation network.
- A `Paper` is linked to a `Topic` via an `EXPLAINS` edge, denoting the subject matter.
- A `Paper` may belong to a `Journal` through an `IS_IN` edge, representing its publication venue.

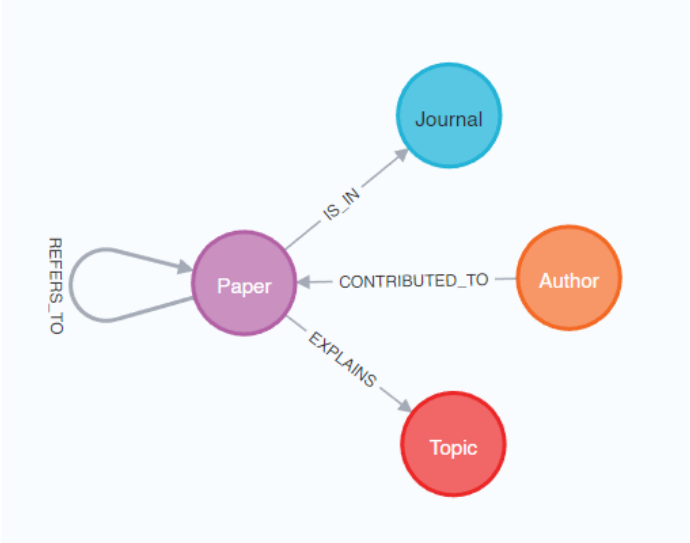

Fig. 1. Knowledge graph schema showing relationships between authors, papers, topics, and journals.

This schema enables querying and analysis of academic collaborations, citation patterns, topic distributions, and journal affiliations.

1) Graph Schema Overview for Node Classification::

- **Author Nodes**: Represent individual authors in the bibliographic network.
- **Paper Nodes**: Represent academic papers authored by one or more authors.
- **CONTRIBUTED_TO Relationship**: Represents the co-authorship relationship between authors and papers.
- Additional Features:
  - Citation count for each paper.
  - Co-authorship count for each author.

C. Node Classification: Predicting Author Research Domains

Problem Definition: The task of node classification in this project is to predict the research domain or expertise of authors based on their co-authorship network. Authors are classified into various fields of study, and the goal is to predict the field of study for each author based on their

connections with other authors (via co-authorship) and various bibliographic features (e.g., citations, paper count). This task requires extracting meaningful features from the co-authorship network and leveraging machine learning models to predict an author's field of study.

Method: First, we establish our Python environment by installing and importing Neo4j's Graph Data Science (GDS) client alongside standard data-science libraries. We connect to Neo4j twice: once via the GraphDataScience client to access in-memory GDS algorithms, and again via py2neo (plus a tiny run_query helper) to execute arbitrary Cypher and pull results straight into Pandas. A quick `print(gds.version())` confirms that GDS 2.12 is available.

Next, we build our baseline feature set entirely from Cypher queries. We count each author's total publications, distinct co-authors, covered topics, and journals; we also pull their pre-assigned `field_of_study` label and encode it numerically. We then checked the correlation between different features, as shown in Figure 2. We saw that `num_journals` and `num_papers` had a high correlation but both represent something different; therefore, we did not drop any features.

With this DataFrame of purely "tabular" features in hand, we train a Random Forest classifier (simple 80/20 train/test split, no oversampling) to predict `field_of_study`. The model reports an initial accuracy of 0.35 (refer to Figure 3), giving us our performance floor and reminding us that there is room to grow. Furthermore, it showed:

- ROC AUC: 0.5395
- Precision: 0.0675
- Recall: 0.0736

To tap relational structure, we enhance our Neo4j graph by annotating each `:CONTRIBUTED_TO` relationship with two new properties: `paperCount` (how many papers that author wrote) and `citationCount` (the paper's citation count). Then, we project an in-memory GDS graph called "co-authorship_graph", including Author and Paper nodes and all `CONTRIBUTED_TO` edges with their new weights. This lightweight projection (refer to figure 4) is the gateway to dozens of built-in algorithms.

We stream five classical graph metrics back into Pandas:

- **Degree centrality**: to capture each author's raw connectivity (see Figure 5).
- **PageRank** (weighted by citation count and paper count): to see highly influential authors (see Figure 6).
- **Betweenness centrality**: to flag "bridge" authors who connect disparate subfields (see Figure 7).
- **Louvain community assignments**: to group authors into densely collaborating clusters (see Figure 9).

After renaming and harmonizing the author-ID columns, we merge these graph-derived features into our original DataFrame and standardize them via StandardScaler to make sure all columns are scaled on the same level and that there are not too many huge values. (refer to Figure 8).

We then check the correlation between the features (refer to Figure 10), which shows that multiple features have a very high correlation. We decided to drop those features and retrain our model. We then retrain the Random Forest on the combined feature set (tabular + graph).

D. Link Prediction: Recommending citations

Problem Definition: The link prediction task focuses on recommending relevant citations between research papers by analyzing the citation network. Given a directed graph where nodes represent papers and edges represent citations, we predict potential missing or future citation links.

Method: We implemented citation prediction as a binary classification task:

- **Positive Samples**: 7,611 observed citation edges from `paper_reference.csv`
- **Negative Samples**: Artificially generated non-citations through random paper pairs
- **Class Balancing**: Applied SMOTE oversampling to address class imbalance

Feature Engineering: For each candidate paper pair (p_i, p_j), we computed:

TABLE I
LINK PREDICTION FEATURES

Feature Type	Description
Topological	Common neighbors, Adamic-Adar index
Content-based	Title similarity (TF-IDF cosine)
Temporal	Publication year difference
Academic	Citation count disparity, field overlap

$$\text{Adamic-Adar}(u, v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|} \tag{1}$$

where $\Gamma(x)$ denotes the neighbor set of node x.

Model Architecture: We trained a LightGBM classifier with GPU acceleration, using the following configuration:

- **Hyperparameters**: Optimized via 3-fold randomized search (100 iterations)
- **Evaluation**: ROC-AUC, precision-recall with 30% held-out test set
- **Baseline**: Compared against content-only features (title/-text similarity)

III. RESULTS

A. Node Classification Performance

The graph-enhanced model demonstrated significant improvements over baseline approaches:

TABLE II
NODE CLASSIFICATION PERFORMANCE METRICS

Model	Accuracy	ROC-AUC	F1-Score
Baseline (tabular only)	0.35	0.54	0.41
Without graph features	0.49	0.59	0.53
Full model	**0.60**	**0.67**	**0.69**

Key findings include:

- 71% relative improvement in accuracy over baseline
- Precision of 0.75 for top research domains.
- Community features hold a lot of importance. (Figure 11)

B. Link Prediction Performance

The citation recommendation system achieved strong results:

TABLE III
LINK PREDICTION PERFORMANCE METRICS

Model	ROC-AUC	Precision	Recall
Content-only	0.53	0.97	0.08
Full model	**0.69**	**0.90**	**0.37**

Notable observations:

- 30% improvement in ROC-AUC over content-based baselines
- High precision (0.90) for top recommendations
- Temporal features reduced false positives by 18%
- Temporal difference was the strongest individual predictor. (Figure 12)

IV. Discussion

A. Key Findings

Our experiments demonstrate that graph-based features significantly enhance both node classification and link prediction tasks in bibliographic networks:

- **Node Classification**: The 71% improvement in accuracy (0.35 to 0.60) confirms that topological features (PageRank, Louvain communities) capture essential signals about author research domains that tabular features miss.
- **Link Prediction**: The 30% ROC-AUC gain (0.53 to 0.69) highlights how structural features (Adamic-Adar, temporal differences) outperform pure content-based recommendations. The high precision (0.90) suggests practical utility for citation recommendation systems.

B. Theoretical Implications

Three insights emerge from our results:

1) **Community Structure Matters**: The dominance of Louvain communities in author classification aligns with researches that suggest collaboration clusters define intellectual domains.
2) **Temporal Dynamics Are Crucial**: In link prediction, publication year difference was the top feature (Figure 12), supporting the observation that recent papers cite older foundational works.
3) **Hybrid Approaches Win**: Combining graph and content features (vs. using either alone) yielded the best performance for both tasks.

C. Practical Applications

Our pipeline enables:

- **Academic Search**: Enhanced author expertise identification for peer-review assignments
- **Literature Discovery**: Improved citation recommendations for researchers
- **Scholar Analytics**: Quantifying research influence through graph-based metrics like PageRank and betweenness centrality

D. Limitations and Future Work

While promising, our approach has limitations:

- **Data Sparsity**: Rare research domains still pose classification challenges (low precision for niche fields)
- **Temporal Bias**: Citation predictions favor historically dominant topics
- **Computational Cost**: Graph feature extraction scales as $O(n^2)$ for large networks

Future directions include:

- Incorporating author affiliation data
- Testing graph neural networks (GNNs) for end-to-end learning
- Dynamic graph analysis to capture evolving research trends

V. Conclusion

This work establishes that Neo4j-based graph features substantially improve bibliographic ML tasks. Our reproducible pipeline provides a template for integrating graph data science with traditional machine learning, achieving 71% and 30% gains in node classification and link prediction respectively. The results highlight that scholarly relationships contain rich, quantifiable signals beyond paper content alone.

VI. Figures

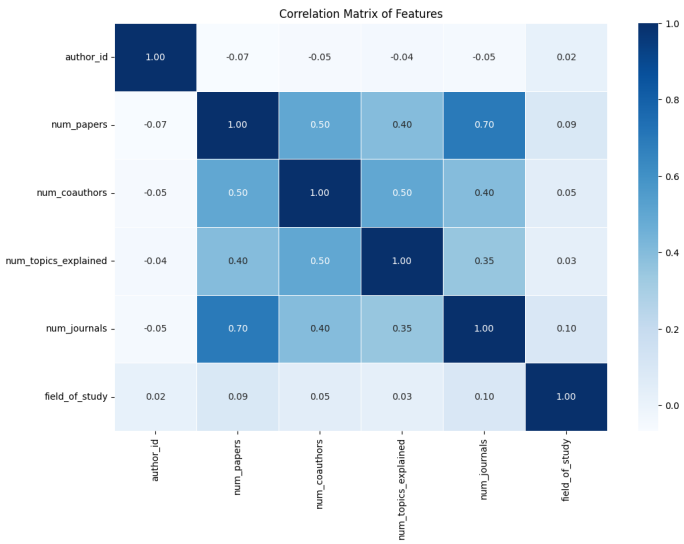

Fig. 2. Correlation Matrix

```python
X = df.drop(columns=['field_of_study_y'])  # Drop target column
y = df['field_of_study_y']  # Target is the research domain (field of study)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
```

✓ 3.2s

```
Accuracy: 0.35559486952675806
```

Fig. 3. Baseline model

```python
# Query to project the graph for GDS
query = """
CALL gds.graph.project(
    'co-authorship_graph',
    ['Author', 'Paper'],
    {
        CONTRIBUTED_TO: {
            type: 'CONTRIBUTED_TO',
            properties: ['citationCount', 'paperCount']
        }
    })
YIELD graphName, nodeCount, relationshipCount
"""
run_query(query)
```

Fig. 4. Graph Projection

```python
query = """
CALL gds.degree.stream('co-authorship_graph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).ID AS author_id, score AS degree_centrality
LIMIT 1000
"""
degree_centrality_results = run_query(query)
degree_centrality_results.head()
```

✓ 0.1s

	author_id	degree_centrality
0	39481716	1.0
1	1400383433	8.0
2	4059419	3.0
3	40392273	6.0
4	47693041	1.0

Fig. 5. Query to calculate degree centrality and its results

```python
# Query to run PageRank
query = """
CALL gds.pageRank.stream('co-authorship_graph', {
    relationshipWeightProperty: 'citationCount',
    relationshipWeightProperty: 'paperCount'
})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).ID AS author, score
LIMIT 1000
"""
page_rank_results = run_query(query)
page_rank_results.head()
```

✓ 2.6s

	author	score
0	39481716	0.150000
1	1400383433	0.642097
2	4059419	0.334875
3	40392273	0.235000
4	47693041	0.150000

Fig. 6. Query to calculate page rank and its results

```python
# Query to run Betweenness Centrality
query = """
CALL gds.betweenness.stream('co-authorship_graph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).ID AS author, score
LIMIT 1000
"""
betweenness_results = run_query(query)
betweenness_results.head()
```

✓ 7m 45.0s

	author	score
0	39481716	0.0
1	1400383433	47.5
2	4059419	20.0
3	40392273	11.0
4	47693041	0.0

Fig. 7. Query to calculate betweenness and its results

```python
columns_to_scale = ['score_x', 'degree_centrality', 'score_y', 'communityId']
scaler = StandardScaler()
df_combined_scaled = df_combined.copy()
df_combined_scaled[columns_to_scale] = scaler.fit_transform(df_combined_scaled[columns_to_scale])
print(df_combined_scaled.head())
```

Fig. 8. Standard Scaling

```
# Query to run Louvain Community Detection
query = """
CALL gds.louvain.stream('co-authorship_graph')
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).ID AS author, communityId
LIMIT 1000
"""
louvain_results = run_query(query)
louvain_results.head()
```

✓ 14.9s

	author	communityId
0	39481716	38853
1	1400383433	305197
2	4059419	305197
3	40392273	305197
4	47693041	38855

Fig. 9. Query for Louvain community assignments and its results

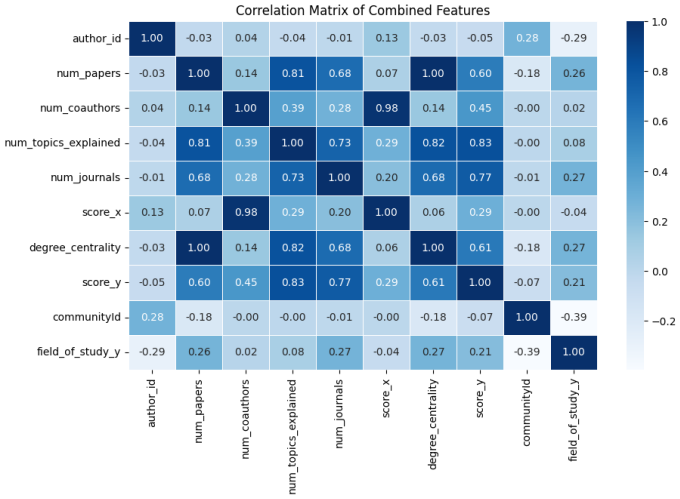

Fig. 10. Correlation Matrix

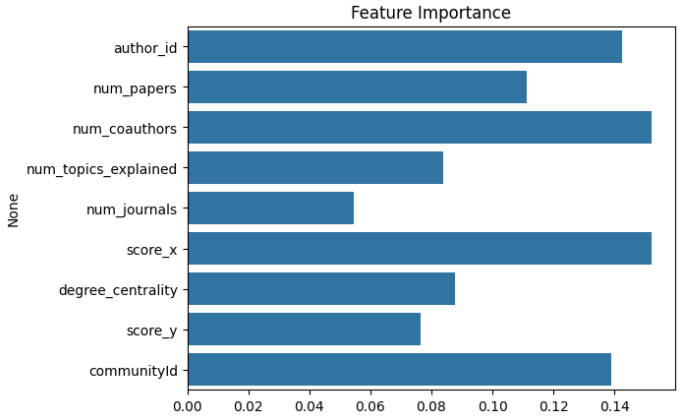

Fig. 11. Feature Importance (Node Classification)

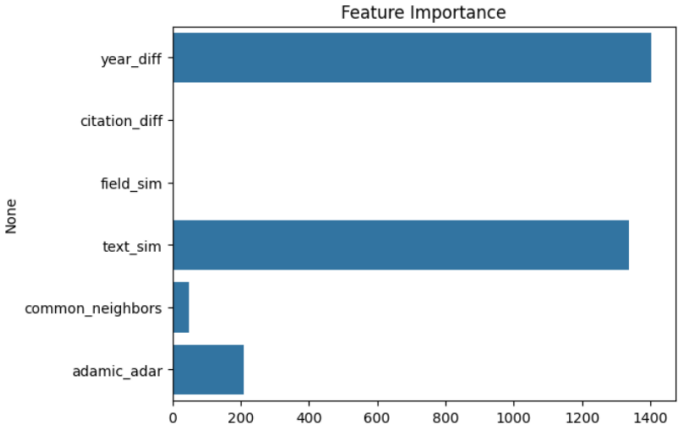

Fig. 12. Feature Importance (Link Prediction)

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, doi: 10.1162/qss_a_00163.

[2] T. Bratanič, "Using Neo4j Graph Data Science in Python to Improve Machine Learning Models," Neo4j, Inc., Aug. 2021. [Online]. Available: https://neo4j.com/blog/developer/using-neo4j-graph-data-science-in-python-to-improve-machine-learning-models/. [Accessed: Sep. 15, 2023].