Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction

We will explore a hands-on example, based on the **Citation Network Dataset**, a research dataset extracted from DBLP, ACM, and MAG. The dataset is described in the paper “ArnetMiner: Extraction and Mining of Academic Social Networks”, by J. Tang et al. The latest version contains 3,079,007 papers, 1,766,547 authors, 9,437,718 author relationships, and 25,166,994 citation relationships.

We’ll be working with a subset focused on articles that appeared in the following publications: Lecture Notes in Computer Science; Communications of the ACM; International Conference on Software Engineering; Advances in Computing and Communications. The subset has 80,289 authors; 51,956 papers; 140,575 author relationships, and 28,706 citation relationships.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction

We will create a co-authors graph based on authors who have collaborated on papers and then predict future collaborations between pairs of authors. We're only interested in collaborations between authors who haven't collaborated before—we're not concerned with multiple collaborations between pairs of authors.

We'll walk through training of a Random Forest Classifier using Python's scikit-learn machine learning library. We will explore feature extraction and prediction using basic graphy features and adding more graph algorithm features extracted using Neo4j.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction - Libraries

py2neo

Neo4j Python library that integrates well with the Python data science ecosystem

pandas

High-performance library for data wrangling outside of a database with easy-to-use data structures and data analysis tools

Scikit-learn

Python's machine learning library

Matplotlib

Python's data visualisation library

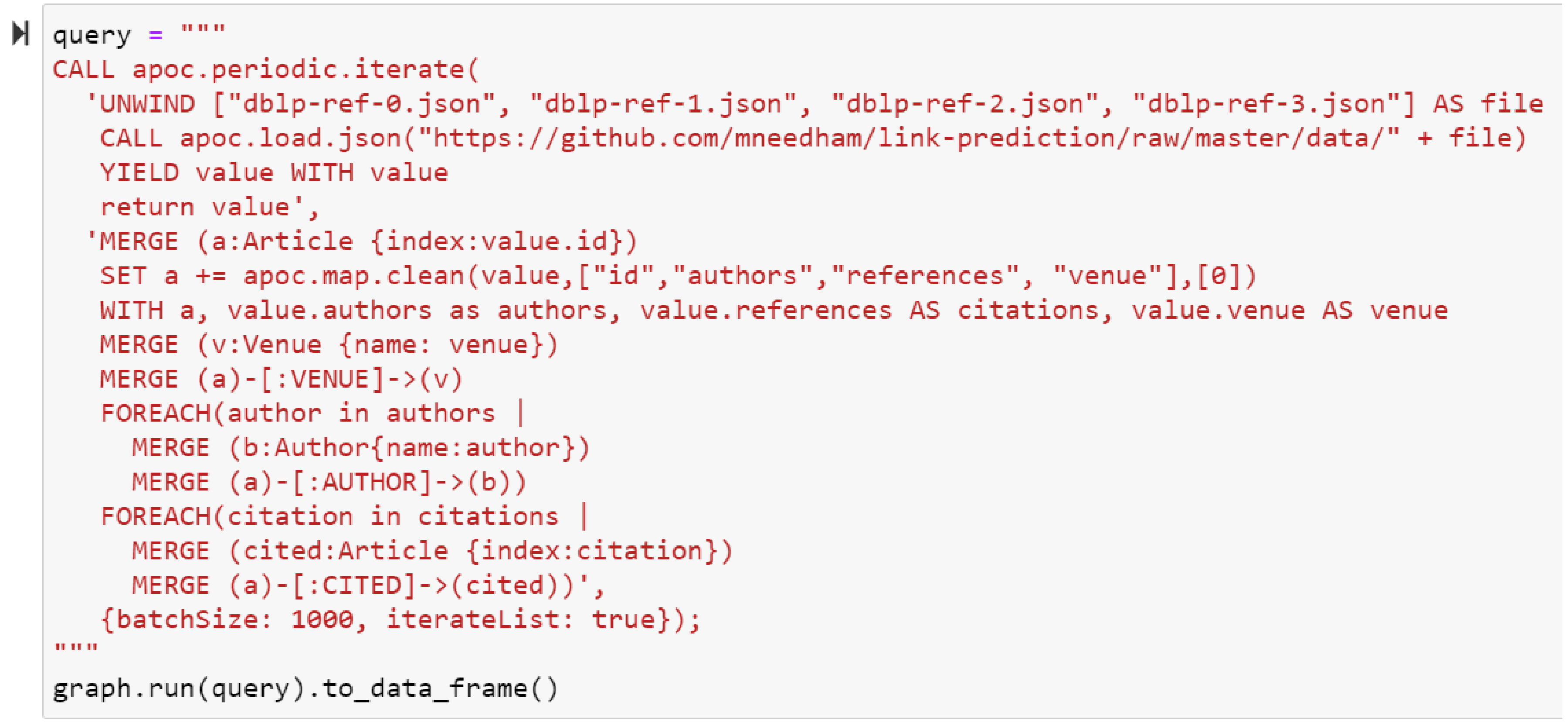
Numpy

Python's array / matrix operations

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Load Data

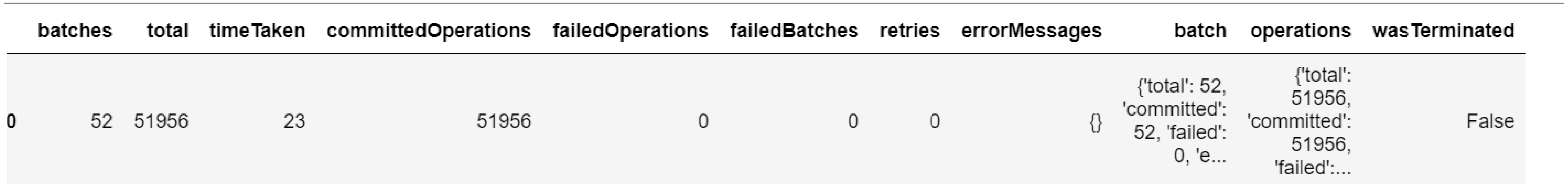
01_DataLoading.ipynb



```
query = """
CALL apoc.periodic.iterate(
  'UNWIND ["dblp-ref-0.json", "dblp-ref-1.json", "dblp-ref-2.json", "dblp-ref-3.json"] AS file
   CALL apoc.load.json("https://github.com/mneedham/link-prediction/raw/master/data/" + file)
   YIELD value WITH value
   return value',
  'MERGE (a:Article {index:value.id})
   SET a += apoc.map.clean(value,["id","authors","references", "venue"],[0])
   WITH a, value.authors as authors, value.references AS citations, value.venue AS venue
   MERGE (v:Venue {name: venue})
   MERGE (a)-[:VENUE]->(v)
   FOREACH(author in authors |
     MERGE (b:Author{name:author})
     MERGE (a)-[:AUTHOR]->(b))
   FOREACH(citation in citations |
     MERGE (cited:Article {index:citation})
     MERGE (a)-[:CITED]->(cited))',
   {batchSize: 1000, iterateList: true});
"""
graph.run(query).to_data_frame()
```

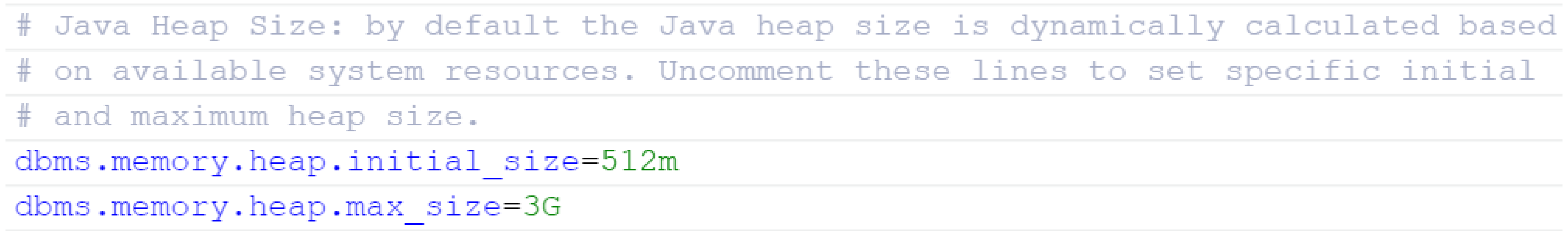
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Load Data



	batches	total	timeTaken	committedOperations	failedOperations	failedBatches	retries	errorMessages	batch	operations	wasTerminated
0	52	51956	23	51956	0	0	0	{}	{'total': 52, 'committed': 52, 'failed': 0, 'e...	{'total': 51956, 'committed': 51956, 'failed':...	False

Note that you will need to increase the heap size on the community edition Citations database in the configuration settings and restart the database.



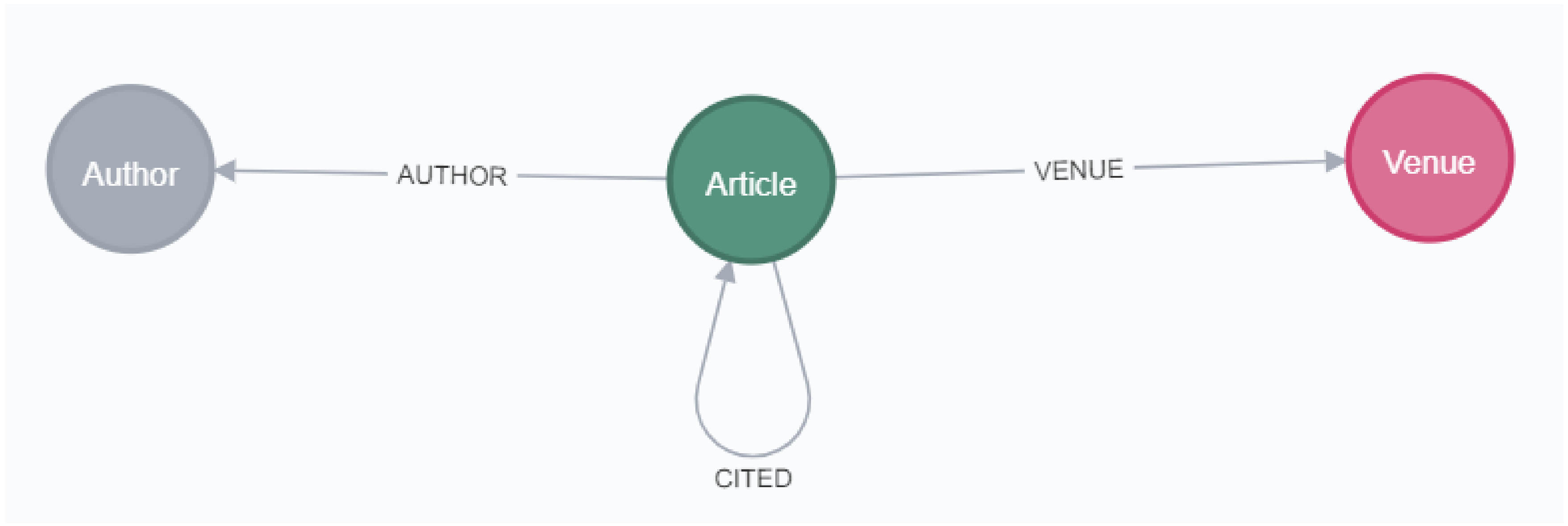
```
# Java Heap Size: by default the Java heap size is dynamically calculated based
# on available system resources. Uncomment these lines to set specific initial
# and maximum heap size.
dbms.memory.heap.initial_size=512m
dbms.memory.heap.max_size=3G
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – EDA – Data Model

02_EDA.ipynb

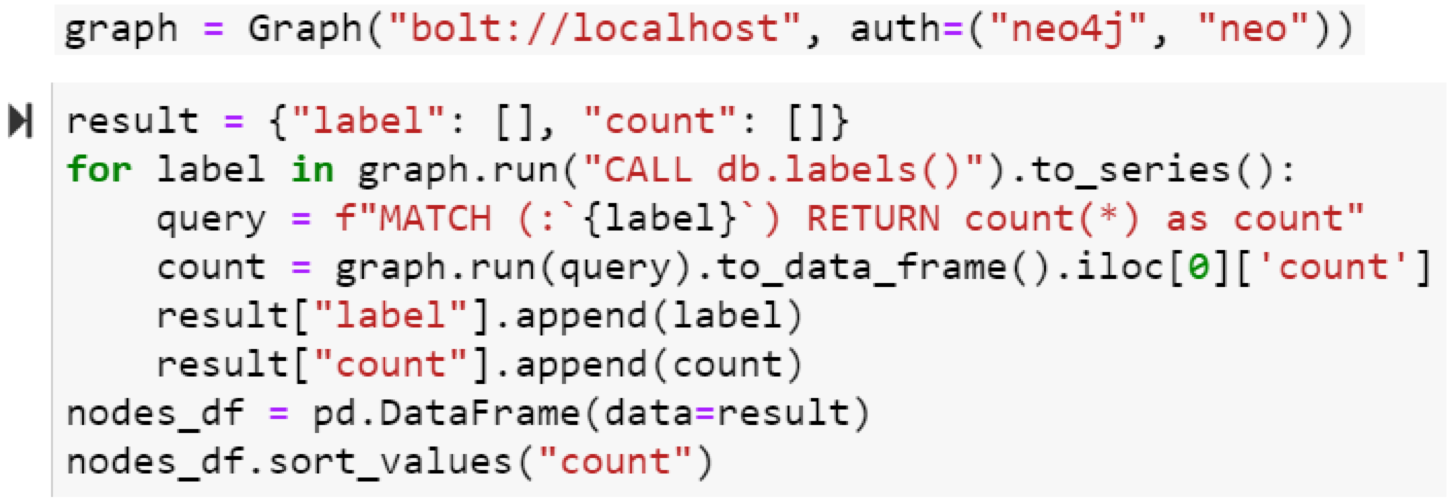
```
graph.run("CALL db.schema.visualization()").data()
```



Graph Algorithms and Machine Learning – Case Study

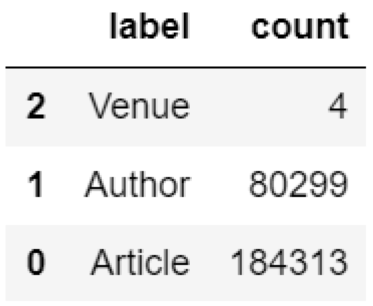
Citations Link Prediction – EDA - Nodes

02_EDA.ipynb



```
graph = Graph("bolt://localhost", auth=("neo4j", "neo"))
```

```
result = {"label": [], "count": []}
for label in graph.run("CALL db.labels()").to_series():
    query = f"MATCH (:`{label}`) RETURN count(*) as count"
    count = graph.run(query).to_data_frame().iloc[0]['count']
    result["label"].append(label)
    result["count"].append(count)
nodes_df = pd.DataFrame(data=result)
nodes_df.sort_values("count")
```



	label	count
2	Venue	4
1	Author	80299
0	Article	184313

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – EDA - Relationships

02_EDA.ipynb

```
graph = Graph("bolt://localhost", auth=("neo4j", "neo"))
```

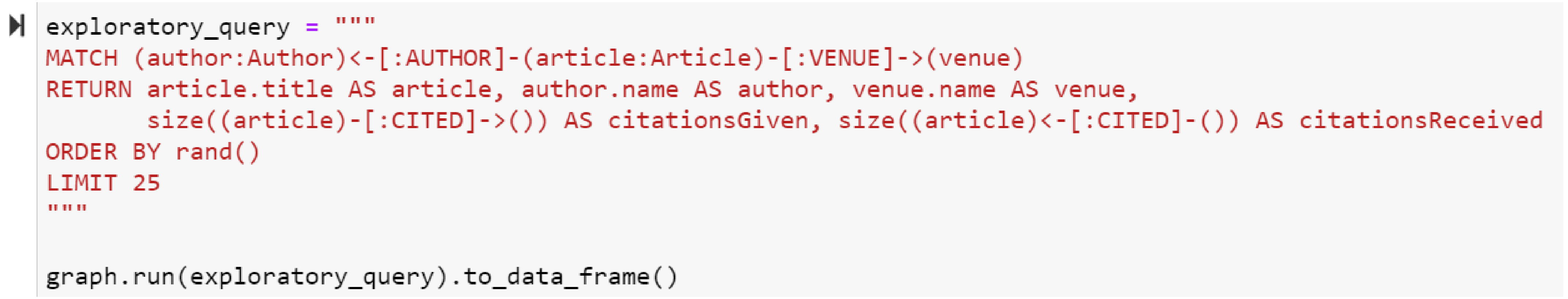
```
result = {"relType": [], "count": []}
for relationship_type in graph.run("CALL db.relationshipTypes()").to_series():
    query = f"MATCH ()-[:`{relationship_type}`]->() RETURN count(*) as count"
    count = graph.run(query).to_data_frame().iloc[0]['count']
    result["relType"].append(relationship_type)
    result["count"].append(count)
rels_df = pd.DataFrame(data=result)
rels_df.sort_values("count")
```

	relType	count
0	VENUE	51956
1	AUTHOR	140575
2	CITED	289908

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – EDA – dataframe output

02_EDA.ipynb

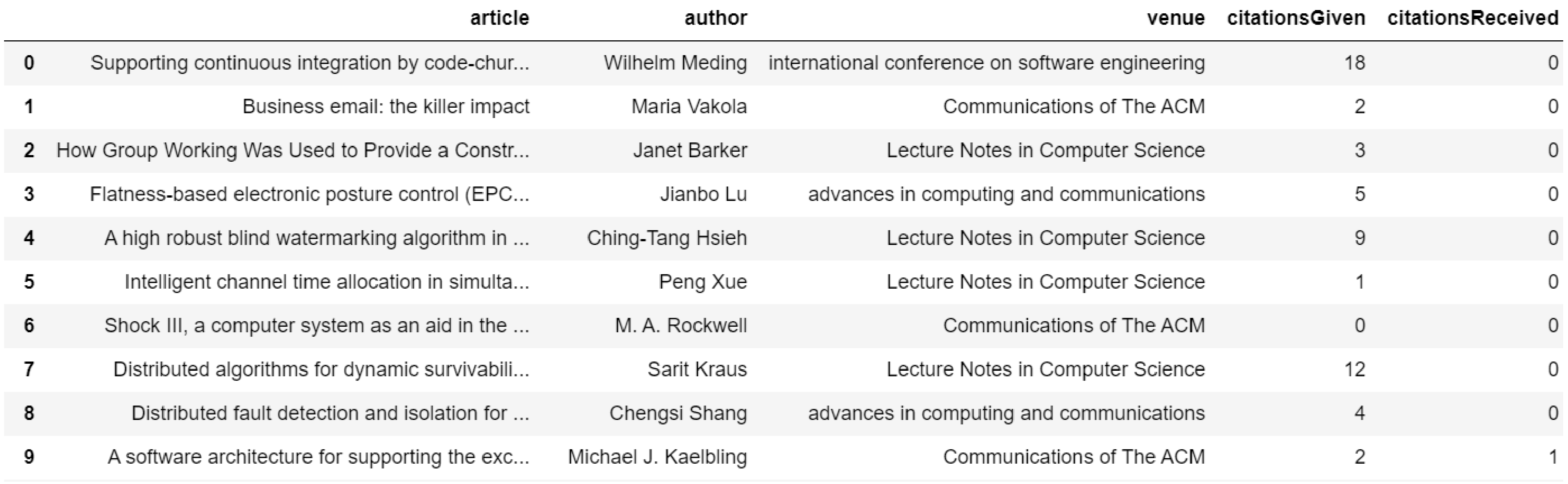


```
exploratory_query = """
MATCH (author:Author)<-[:AUTHOR]-(article:Article)-[:VENUE]->(venue)
RETURN article.title AS article, author.name AS author, venue.name AS venue,
       size((article)-[:CITED]->()) AS citationsGiven, size((article)<-[:CITED]-()) AS citationsReceived
ORDER BY rand()
LIMIT 25
"""

graph.run(exploratory_query).to_data_frame()
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – EDA – dataframe output

02_EDA.ipynb (sample of output)

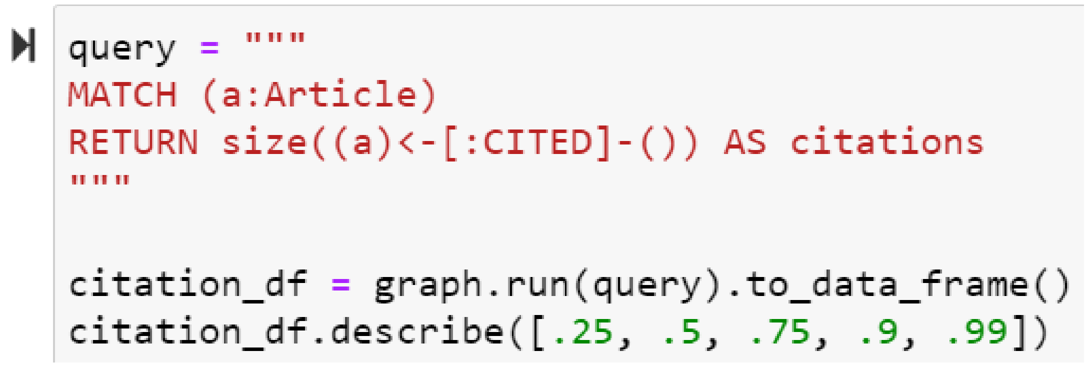


	article	author	venue	citationsGiven	citationsReceived
0	Supporting continuous integration by code-chur...	Wilhelm Meding	international conference on software engineering	18	0
1	Business email: the killer impact	Maria Vakola	Communications of The ACM	2	0
2	How Group Working Was Used to Provide a Constr...	Janet Barker	Lecture Notes in Computer Science	3	0
3	Flatness-based electronic posture control (EPC...	Jianbo Lu	advances in computing and communications	5	0
4	A high robust blind watermarking algorithm in ...	Ching-Tang Hsieh	Lecture Notes in Computer Science	9	0
5	Intelligent channel time allocation in simulta...	Peng Xue	Lecture Notes in Computer Science	1	0
6	Shock III, a computer system as an aid in the ...	M. A. Rockwell	Communications of The ACM	0	0
7	Distributed algorithms for dynamic survivabili...	Sarit Kraus	Lecture Notes in Computer Science	12	0
8	Distributed fault detection and isolation for ...	Chengsi Shang	advances in computing and communications	4	0
9	A software architecture for supporting the exc...	Michael J. Kaelbling	Communications of The ACM	2	1

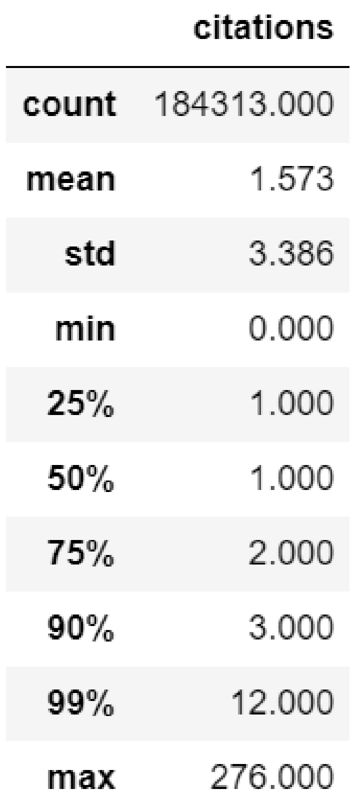
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – EDA – Statistics

02_EDA.ipynb



```
query = """
MATCH (a:Article)
RETURN size((a)<-[:CITED]-()) AS citations
"""

citation_df = graph.run(query).to_data_frame()
citation_df.describe([.25, .5, .75, .9, .99])
```



	citations
count	184313.000
mean	1.573
std	3.386
min	0.000
25%	1.000
50%	1.000
75%	2.000
90%	3.000
99%	12.000
max	276.000

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Co-Authorship sub-graph

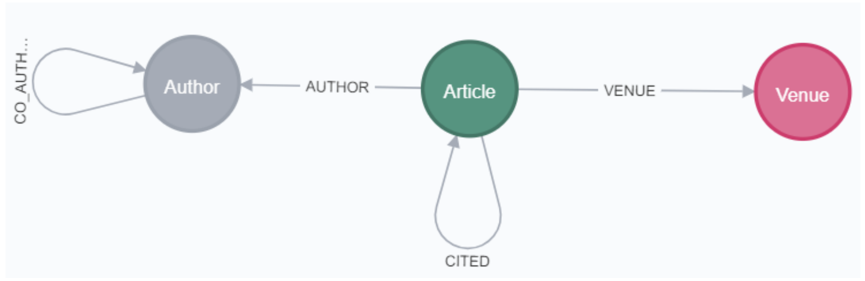
03_Prediction.ipynb

We add more information the can be inferred from relationships to help with predictions. As we want to predict future collaborations between authors, we start by creating a co-authorship graph. The year property that is set on the CO_AUTHOR relationship in the query is the earliest year when those two authors collaborated. We're only interested in the first time that a pair of authors have collaborated—subsequent collaborations aren't relevant.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Co-Authorship sub-graph

03_Prediction.ipynb



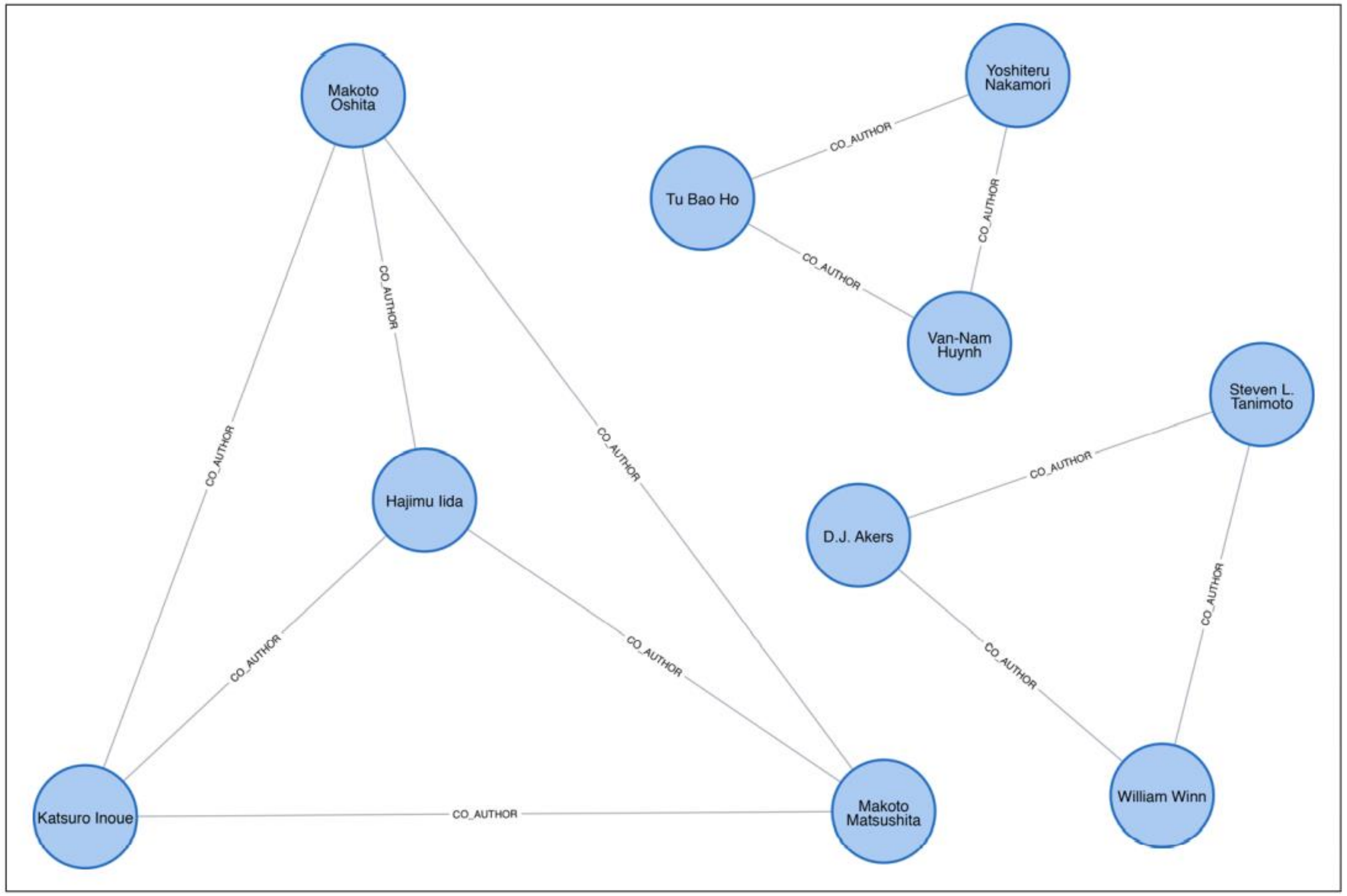
```
query = """
MATCH (a1)<-[:AUTHOR]-(paper)-[:AUTHOR]->(a2:Author)
WITH a1, a2, paper
ORDER BY a1, paper.year
WITH a1, a2, collect(paper)[0].year AS year, count(*) AS collaborations
MERGE (a1)-[coauthor:CO_AUTHOR {year: year}]-(a2)
SET coauthor.collaborations = collaborations;
"""

graph.run(query).stats()
```

```
constraints_added: 0
constraints_removed: 0
contains_updates: True
indexes_added: 0
indexes_removed: 0
labels_added: 0
labels_removed: 0
nodes_created: 0
nodes_deleted: 0
properties_set: 465672
relationships_created: 155224
relationships_deleted: 0
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Co-Authorship sub-graph

Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019



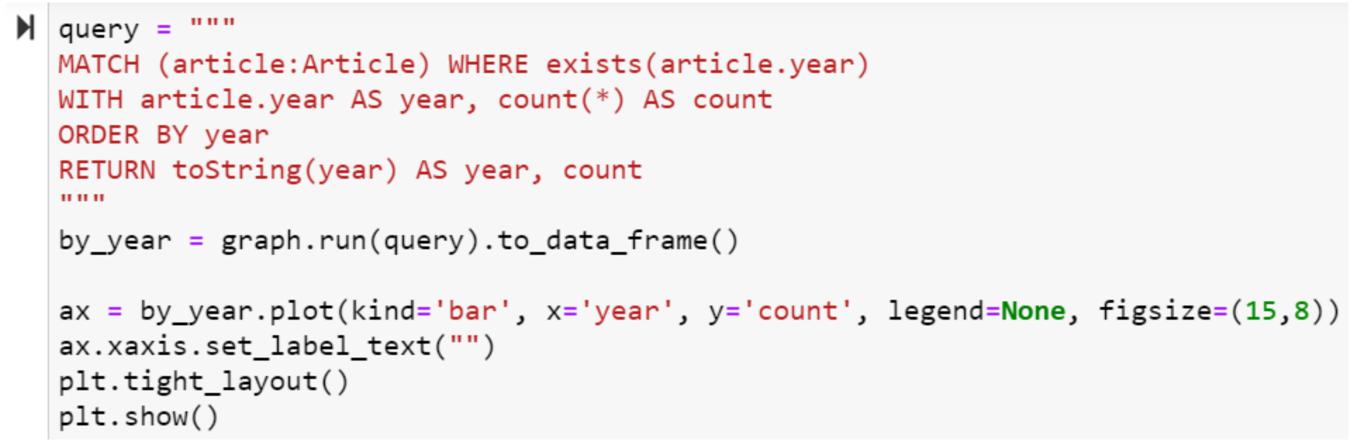
Each node in this sub-graph represents one author and the edges between them are CO_AUTHOR relationships, so we have four authors that have all collaborated with each other on the left, and then on the right two examples of three authors who have collaborated.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Train and Test data

03_Prediction.ipynb

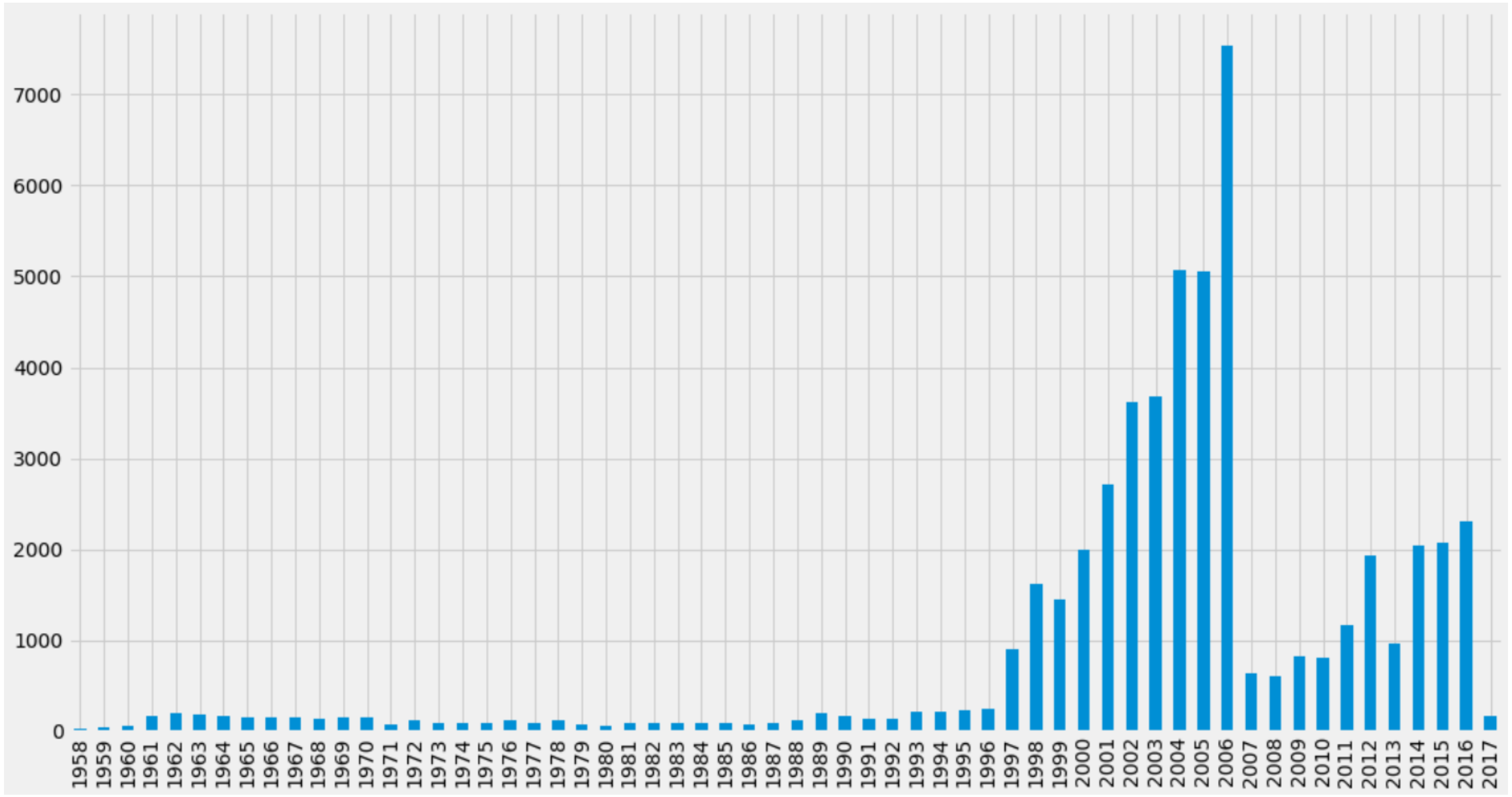
With link prediction problems we want to try and predict the future creation of links. This dataset works well for that because we have dates on the articles that we can use to split our data. We need to work out which year we'll use to define the training/test split. We'll train our model on everything before that year and then test it on the links created after that date. We start by finding out when the articles were published.



```
query = """
MATCH (article:Article) WHERE exists(article.year)
WITH article.year AS year, count(*) AS count
ORDER BY year
RETURN toString(year) AS year, count
"""
by_year = graph.run(query).to_data_frame()

ax = by_year.plot(kind='bar', x='year', y='count', legend=None, figsize=(15,8))
ax.xaxis.set_label_text("")
plt.tight_layout()
plt.show()
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Train and Test data



Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Train and Test data

Few articles were published before 1997, many were published between 2001 and 2006, before a dip and then a gradual climb since 2011 (excluding 2013). It looks like 2006 could be a good year to split our data for training our model and making predictions. Let’s check how many papers were published before that year and how many during and after. True (below) means a paper was published before 2006. 60% of the papers were published before 2006 and 40% during or after 2006. This is a fairly balanced split of data for our training and testing.

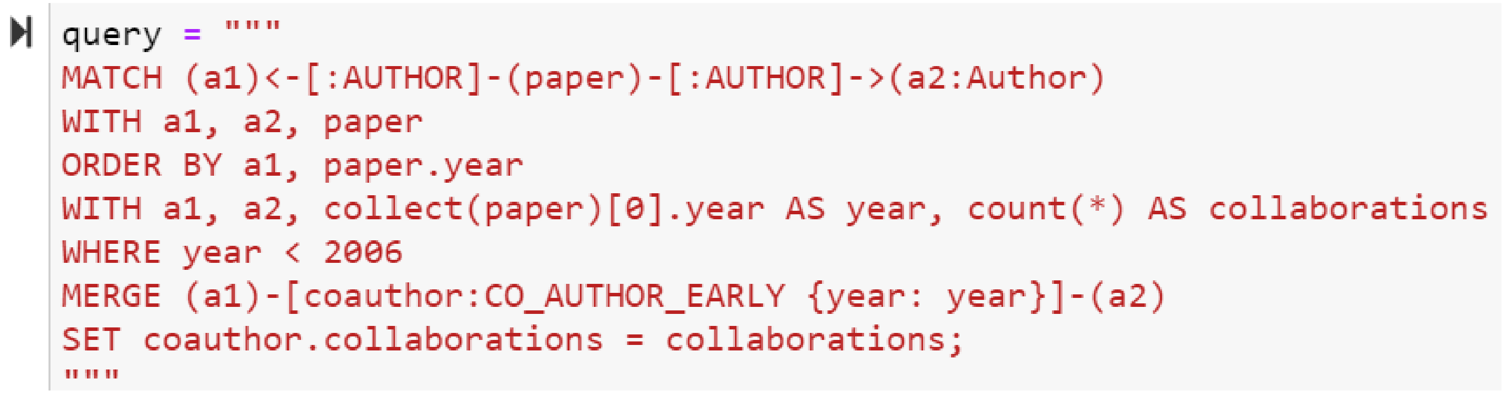
```
MATCH (article:Article)
RETURN article.year < 2006 AS training, count(*) AS count
```

training	count
false	21059
true	30897

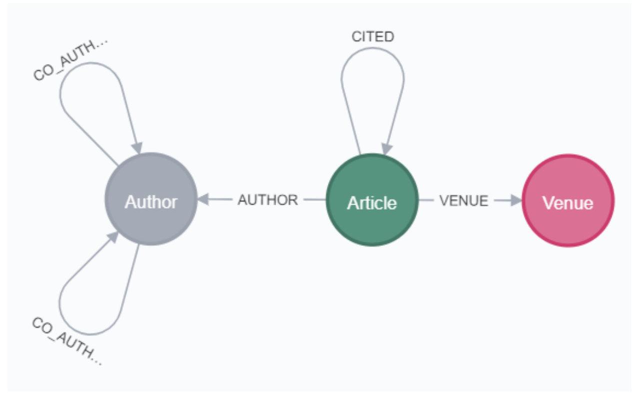
Graph Algorithms and Machine Learning – Case Study

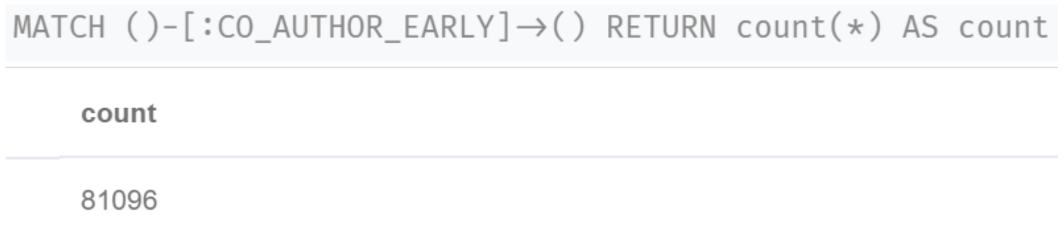
Citations Link Prediction – Train and Test data

We'll create a CO_AUTHOR_EARLY relationship between pairs of authors whose first collaboration was before 2006.



```
query = """
MATCH (a1)<-[:AUTHOR]-(paper)-[:AUTHOR]->(a2:Author)
WITH a1, a2, paper
ORDER BY a1, paper.year
WITH a1, a2, collect(paper)[0].year AS year, count(*) AS collaborations
WHERE year < 2006
MERGE (a1)-[coauthor:CO_AUTHOR_EARLY {year: year}]-(a2)
SET coauthor.collaborations = collaborations;
"""
```





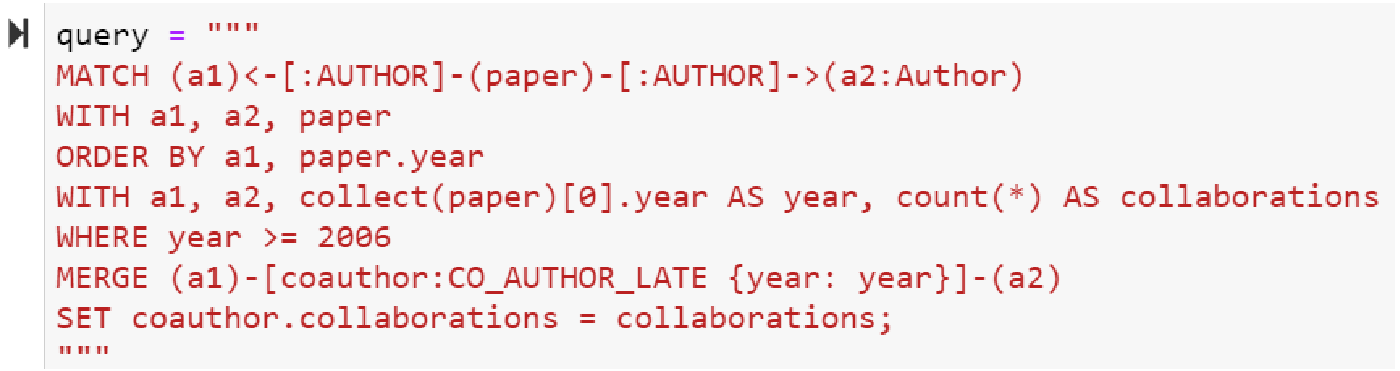
```
MATCH ()-[:CO_AUTHOR_EARLY]→() RETURN count(*) AS count
```

count
81096

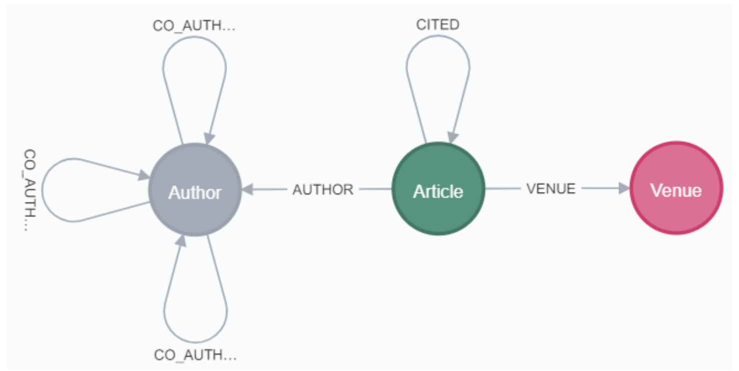
Graph Algorithms and Machine Learning – Case Study

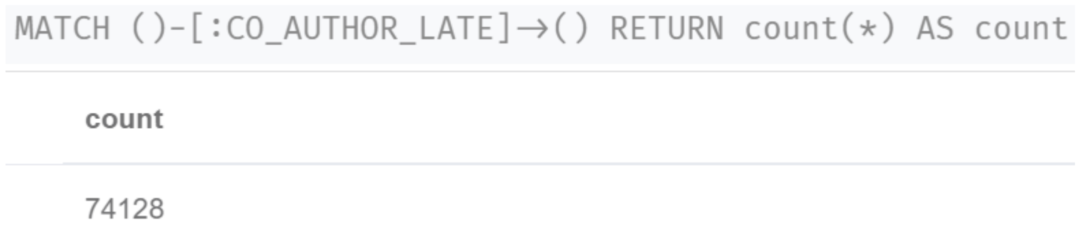
Citations Link Prediction – Train and Test data

We'll create a CO_AUTHOR_LATE relationship between pairs of authors whose first collaboration was after on or after 2006.



```
query = """
MATCH (a1)<-[:AUTHOR]-(paper)-[:AUTHOR]->(a2:Author)
WITH a1, a2, paper
ORDER BY a1, paper.year
WITH a1, a2, collect(paper)[0].year AS year, count(*) AS collaborations
WHERE year >= 2006
MERGE (a1)-[coauthor:CO_AUTHOR_LATE {year: year}]-(a2)
SET coauthor.collaborations = collaborations;
"""
```





```
MATCH ()-[:CO_AUTHOR_LATE]→() RETURN count(*) AS count
```

count
74128

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Balancing and Splitting Data

The pairs of nodes with CO_AUTHOR_EARLY and CO_AUTHOR_LATE relationships between them will act as our positive examples, but we'll also need to create some negative examples. Most real-world networks are sparse, with concentrations of relationships, and this graph is no different. The number of examples where two nodes do not have a relationship is much larger than the number that do have a relationship.

If we query CO_AUTHOR_EARLY data, we find there are 45,018 authors with that type of relationship but only 81,096 relationships between authors. That might not sound imbalanced, but it is: the potential maximum number of relationships that our graph could have is (45018 * 45017) / 2 = 1,013,287,653, which means there are a lot of negative examples (no links). If we use all the negative examples to train a model, we'd have a severe class imbalance problem. A model could achieve extremely high accuracy by predicting that every pair of nodes doesn't have a relationship.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Balancing and Splitting Data

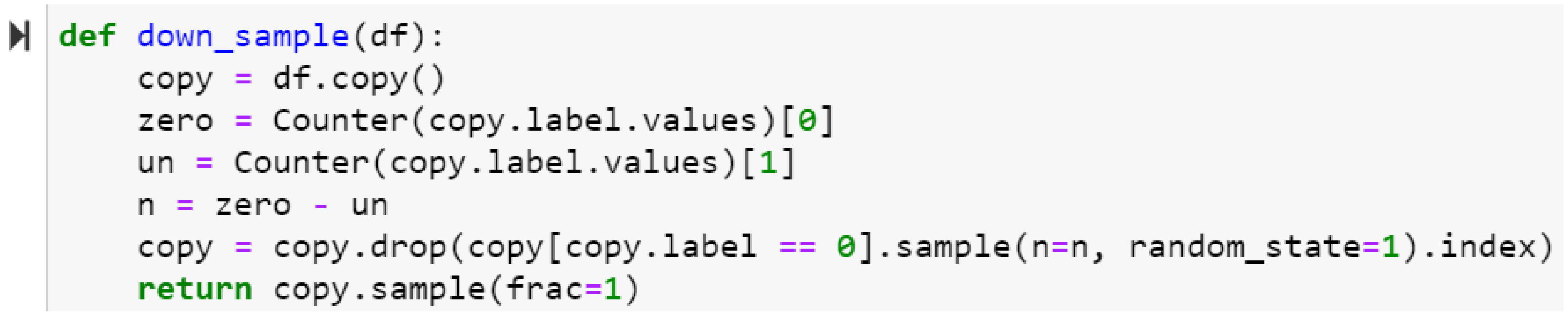
In their paper "New Perspectives and Methods in Link Prediction", R. Lichtenwalter, J. Lussier, and N. Chawla describe several methods to address this challenge. One of these approaches is to build negative examples by finding nodes within the neighbour-hood that we aren't currently connected to. We will build negative examples by finding pairs of nodes that are a mix of between two and three hops away from each other, excluding those pairs that already have a relationship. We'll then down-sample those pairs of nodes so that we have an equal number of positive and negative examples.

We have 314,248 pairs of nodes that don't have a relationship between each other at a distance of two hops. If we increase the distance to three hops, we have 967,677 pairs of nodes.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Balancing and Splitting Data

The following function will be used to down-sample the negative examples. This function works out the difference between the number of positive and negative examples, and then samples the negative examples so that there are equal numbers.



```
def down_sample(df):
    copy = df.copy()
    zero = Counter(copy.label.values)[0]
    un = Counter(copy.label.values)[1]
    n = zero - un
    copy = copy.drop(copy[copy.label == 0].sample(n=n, random_state=1).index)
    return copy.sample(frac=1)
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Balancing and Splitting Data

We can run the following code to build a training set with balanced positive and negative examples.

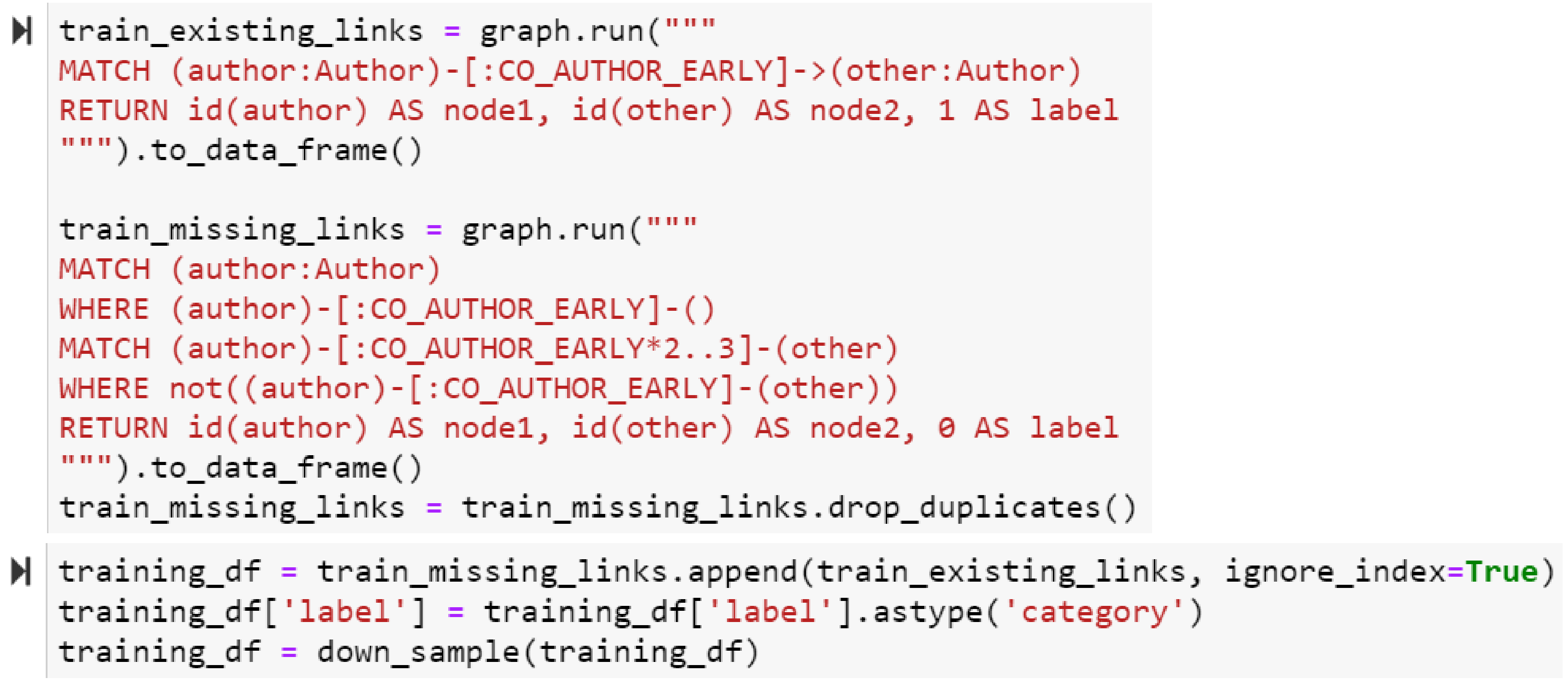
```
train_existing_links = graph.run("""
MATCH (author:Author)-[:CO_AUTHOR_EARLY]->(other:Author)
RETURN id(author) AS node1, id(other) AS node2, 1 AS label
""").to_data_frame()

train_missing_links = graph.run("""
MATCH (author:Author)
WHERE (author)-[:CO_AUTHOR_EARLY]-()
MATCH (author)-[:CO_AUTHOR_EARLY*2..3]-(other)
WHERE not((author)-[:CO_AUTHOR_EARLY]-(other))
RETURN id(author) AS node1, id(other) AS node2, 0 AS label
""").to_data_frame()
train_missing_links = train_missing_links.drop_duplicates()
```

```
training_df = train_missing_links.append(train_existing_links, ignore_index=True)
training_df['label'] = training_df['label'].astype('category')
training_df = down_sample(training_df)
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Balancing and Splitting Data

We can run the following code to build a training set with balanced positive and negative examples.



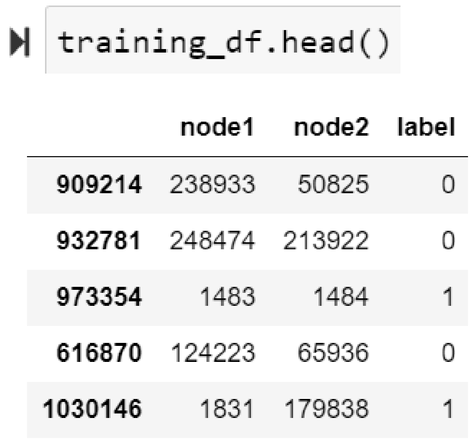
```
train_existing_links = graph.run("""
MATCH (author:Author)-[:CO_AUTHOR_EARLY]->(other:Author)
RETURN id(author) AS node1, id(other) AS node2, 1 AS label
""").to_data_frame()

train_missing_links = graph.run("""
MATCH (author:Author)
WHERE (author)-[:CO_AUTHOR_EARLY]-()
MATCH (author)-[:CO_AUTHOR_EARLY*2..3]-(other)
WHERE not((author)-[:CO_AUTHOR_EARLY]-(other))
RETURN id(author) AS node1, id(other) AS node2, 0 AS label
""").to_data_frame()
train_missing_links = train_missing_links.drop_duplicates()
```

```
training_df = train_missing_links.append(train_existing_links, ignore_index=True)
training_df['label'] = training_df['label'].astype('category')
training_df = down_sample(training_df)
```

Graph Algorithms and Machine Learning – Case Study

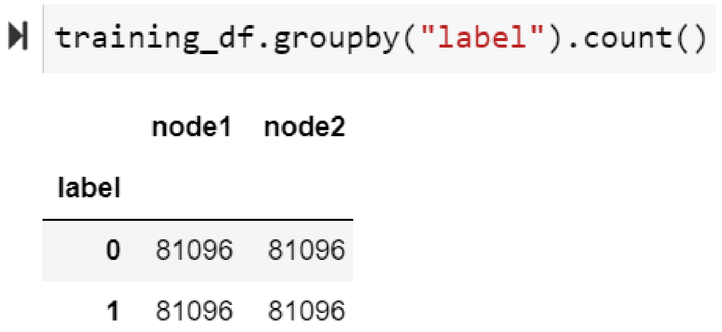
Citations Link Prediction – Balancing and Splitting Data

The results show us a list of node pairs and whether they have a co-author relationship; for example, nodes 1483 and 1484 have a 1 label, indicating a collaboration. The classes are balanced.



```
training_df.head()
```

```
          node1   node2  label
909214   238933   50825      0
932781   248474  213922      0
973354     1483    1484      1
616870   124223   65936      0
1030146    1831  179838      1
```



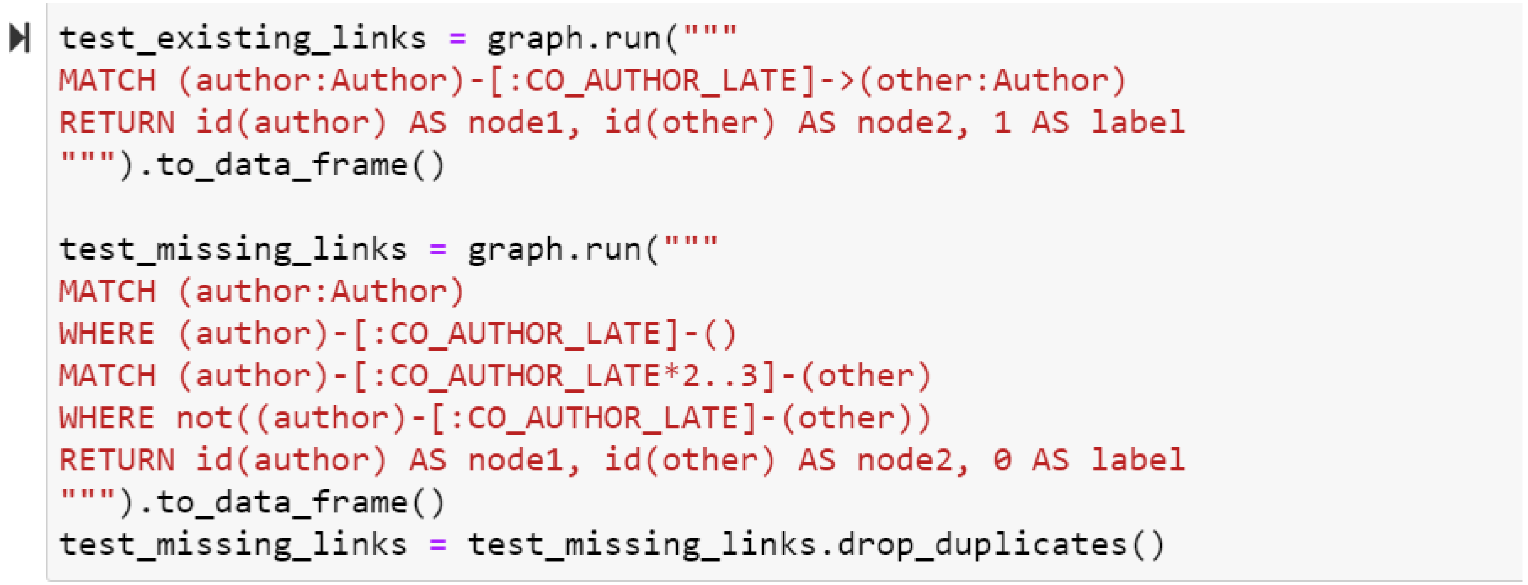
```
training_df.groupby("label").count()
```

```
       node1   node2
label
    0  81096   81096
    1  81096   81096
```

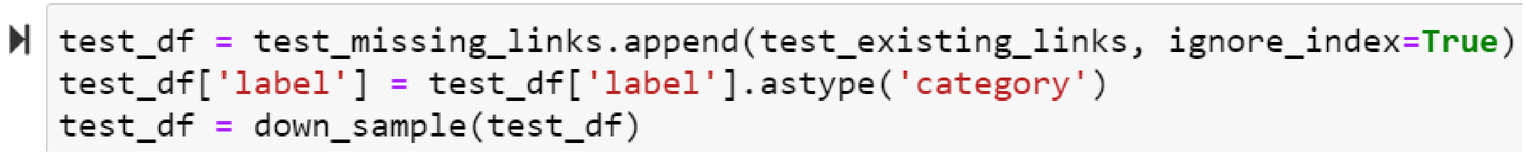
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Balancing and Splitting Data

Now we need to do the same for the test set. The following code will build a test set with balanced positive and negative examples.



```
test_existing_links = graph.run("""
MATCH (author:Author)-[:CO_AUTHOR_LATE]->(other:Author)
RETURN id(author) AS node1, id(other) AS node2, 1 AS label
""").to_data_frame()

test_missing_links = graph.run("""
MATCH (author:Author)
WHERE (author)-[:CO_AUTHOR_LATE]-()
MATCH (author)-[:CO_AUTHOR_LATE*2..3]-(other)
WHERE not((author)-[:CO_AUTHOR_LATE]-(other))
RETURN id(author) AS node1, id(other) AS node2, 0 AS label
""").to_data_frame()
test_missing_links = test_missing_links.drop_duplicates()
```

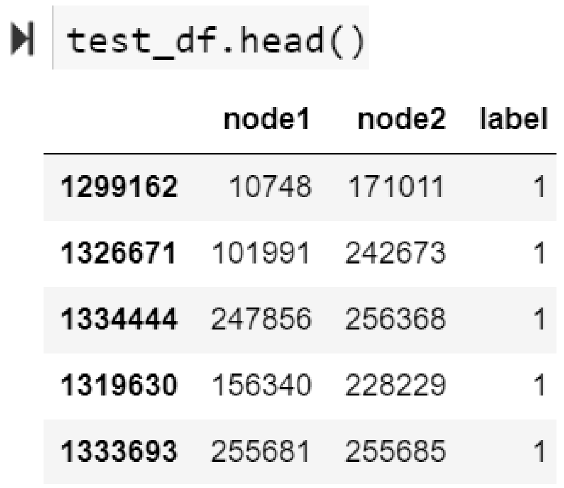


```
test_df = test_missing_links.append(test_existing_links, ignore_index=True)
test_df['label'] = test_df['label'].astype('category')
test_df = down_sample(test_df)
```

Graph Algorithms and Machine Learning – Case Study

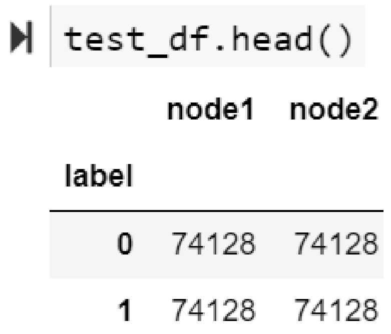
Citations Link Prediction – Balancing and Splitting Data

We have balanced both training and test datasets.



```
test_df.head()
```

	node1	node2	label
1299162	10748	171011	1
1326671	101991	242673	1
1334444	247856	256368	1
1319630	156340	228229	1
1333693	255681	255685	1



```
test_df.head()
```

	node1	node2
label		
0	74128	74128
1	74128	74128

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – How to Predict Missing Links?

We need to start with some basic assumptions about what elements in our data might predict whether two authors will become co-authors at a later date. Our hypothesis would vary by domain and problem, but in this case, we believe the most predictive features will be related to communities. We'll begin with the assumption that the following elements increase the probability that authors become co-authors:

- More co-authors in common
- Potential triadic relationships between authors
- Authors with more relationships
- Authors in the same community
- Authors in the same, tighter community

Graph Algorithms and Machine Learning – Case Study

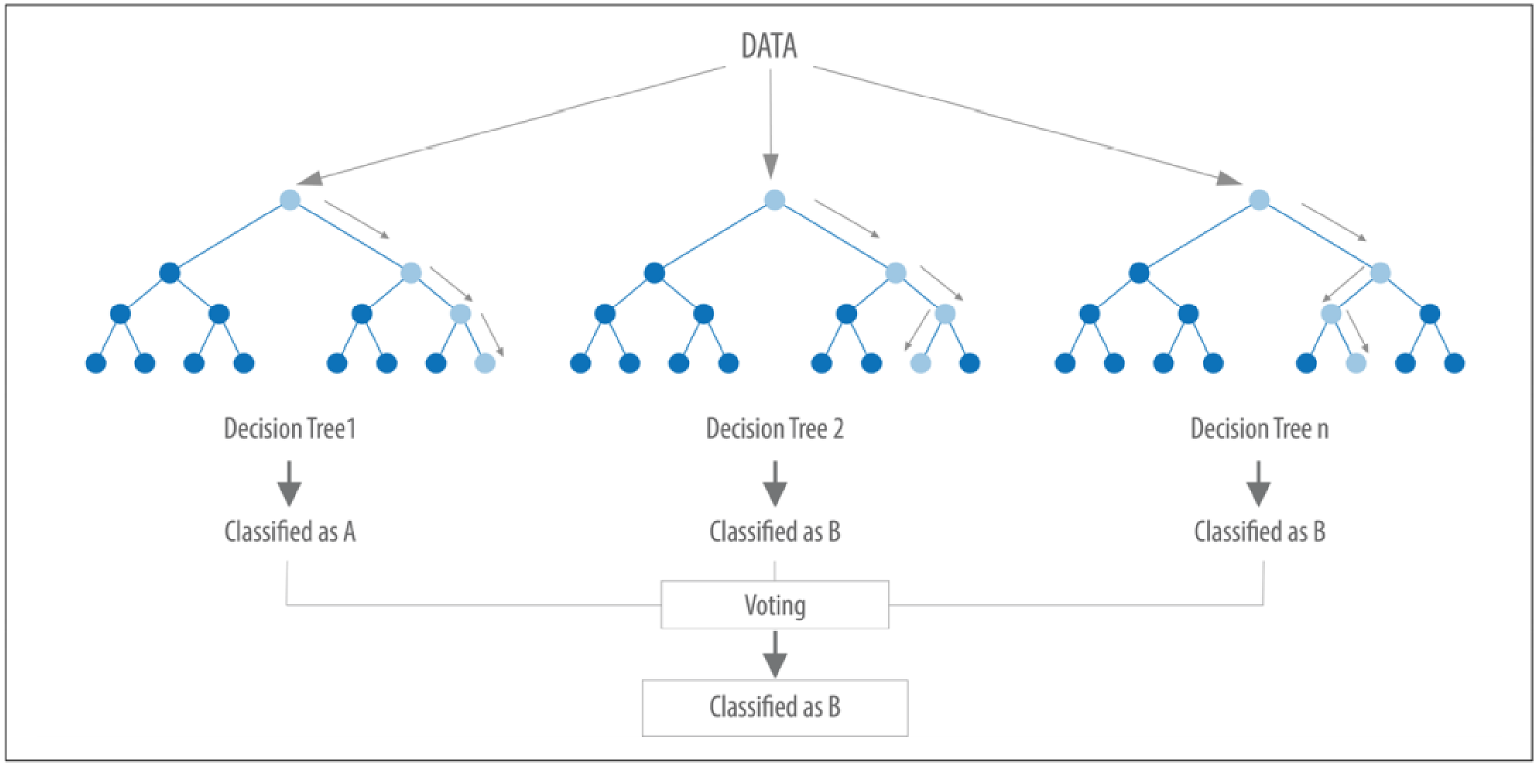
Citations Link Prediction – How to Predict Missing Links?

We'll build graph features based on our assumptions and use those to train a binary classifier. Binary classification is a type of ML with the task of predicting which of two predefined groups an element belongs to based on a rule. We're using the classifier for the task of predicting whether a pair of authors will have a link or not, based on a classification rule. For our examples, a value of 1 means there is a link (co-authorship), and a value of 0 means there isn't a link (no co-authorship).

We will implement the binary classifier as a random forest. A random forest is a supervised ensemble learning method for classification, regression, and other tasks, as illustrated in Figure (next slide). Our random forest classifier will take the results from the multiple decision trees we train and then use voting to predict a classification—in our example, whether there is a link (co-authorship) or not.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – How to Predict Missing Links?



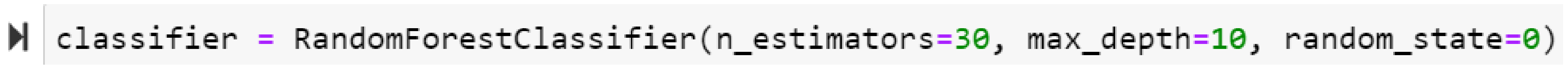
A random forest builds a collection of decision trees and then aggregates results for a majority vote (for classification) or an average value (for regression).

Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Machine Learning Pipeline

We will create our machine learning pipeline based on a random forest classifier in scikit-learn. This method is well suited as our dataset will be comprised of a mix of strong and weak features. While the weak features will sometimes be helpful, the random forest method will ensure we don’t create a model that only fits our training data (overfits on the training data).



```
classifier = RandomForestClassifier(n_estimators=30, max_depth=10, random_state=0)
```

n_estimators The number of decision trees that form the random forest
max_depth The maximum depth of the decision trees

The hyperparameters above are selected to optimise performance by experimentation. To create the ML pipeline, we also add the list of graph engineered features to the dataframes for use during model training.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

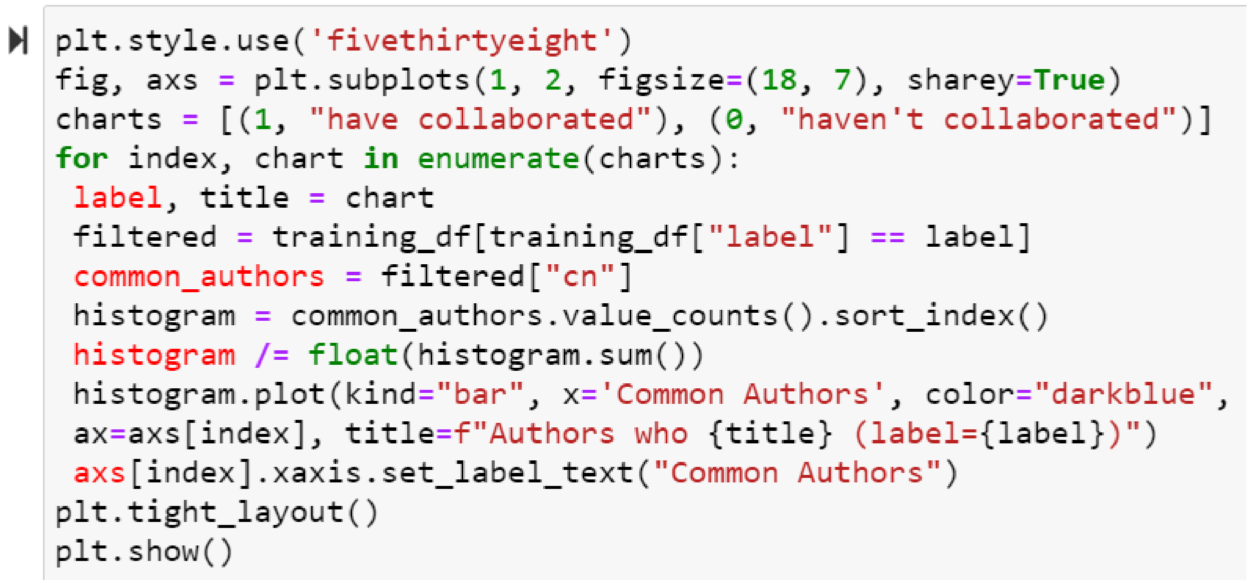
We start by creating a simple model that tries to predict whether two authors will have a future collaboration based on features extracted from common authors, preferential attachment, and the total union of neighbours:

- **Common authors** Finds the number of potential triangles between two authors. This captures the idea that two authors who have co-authors in common may be introduced and collaborate in the future.
- **Preferential attachment** Produces a score for each pair of authors by multiplying the number of co-authors each has. The intuition is that authors are more likely to collaborate with someone who already co-authors a lot of papers.
- **Total union of neighbours** Finds the total number of co-authors that each author has, minus the duplicates.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

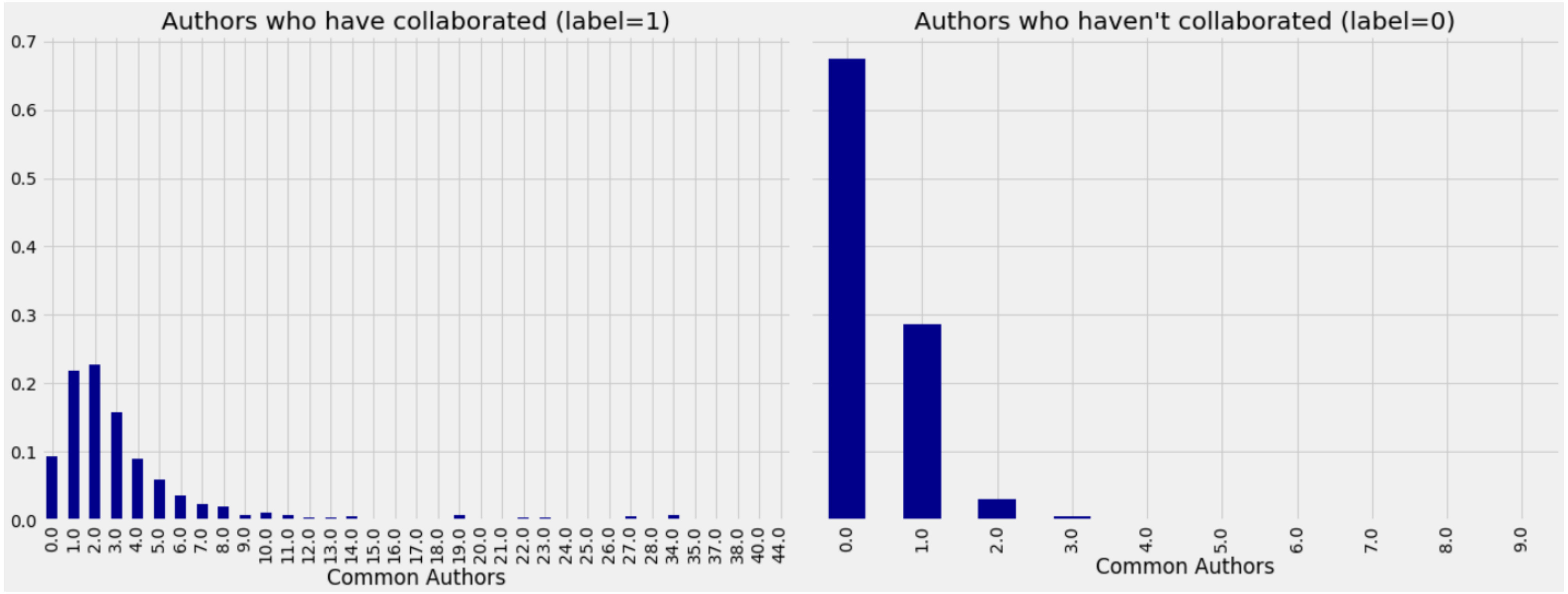
We can use the following code to visually explore the distributions of common authors.



```python
plt.style.use('fivethirtyeight')
fig, axs = plt.subplots(1, 2, figsize=(18, 7), sharey=True)
charts = [(1, "have collaborated"), (0, "haven't collaborated")]
for index, chart in enumerate(charts):
 label, title = chart
 filtered = training_df[training_df["label"] == label]
 common_authors = filtered["cn"]
 histogram = common_authors.value_counts().sort_index()
 histogram /= float(histogram.sum())
 histogram.plot(kind="bar", x='Common Authors', color="darkblue",
 ax=axs[index], title=f"Authors who {title} (label={label})")
 axs[index].xaxis.set_label_text("Common Authors")
plt.tight_layout()
plt.show()
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

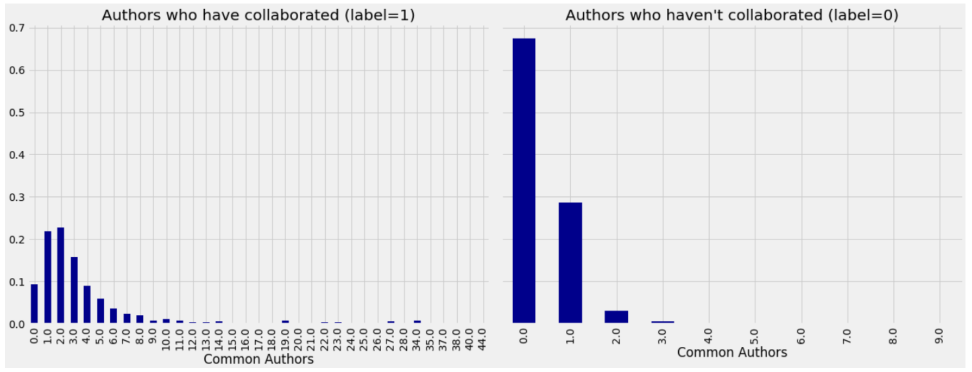


Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

On the left we see the frequency of common Authors when authors have collaborated, and on the right we see the frequency of common Authors when they haven't. For those who haven't collaborated (right side) the maximum number of common authors is 9, but 95% of the values are 1 or 0. It's not surprising that of the people who have not collaborated on a paper, most also do not have many other co-authors in common. For those who have collaborated (left side), 70% have less than five co-authors in common, with a spike between one and two other co-authors.

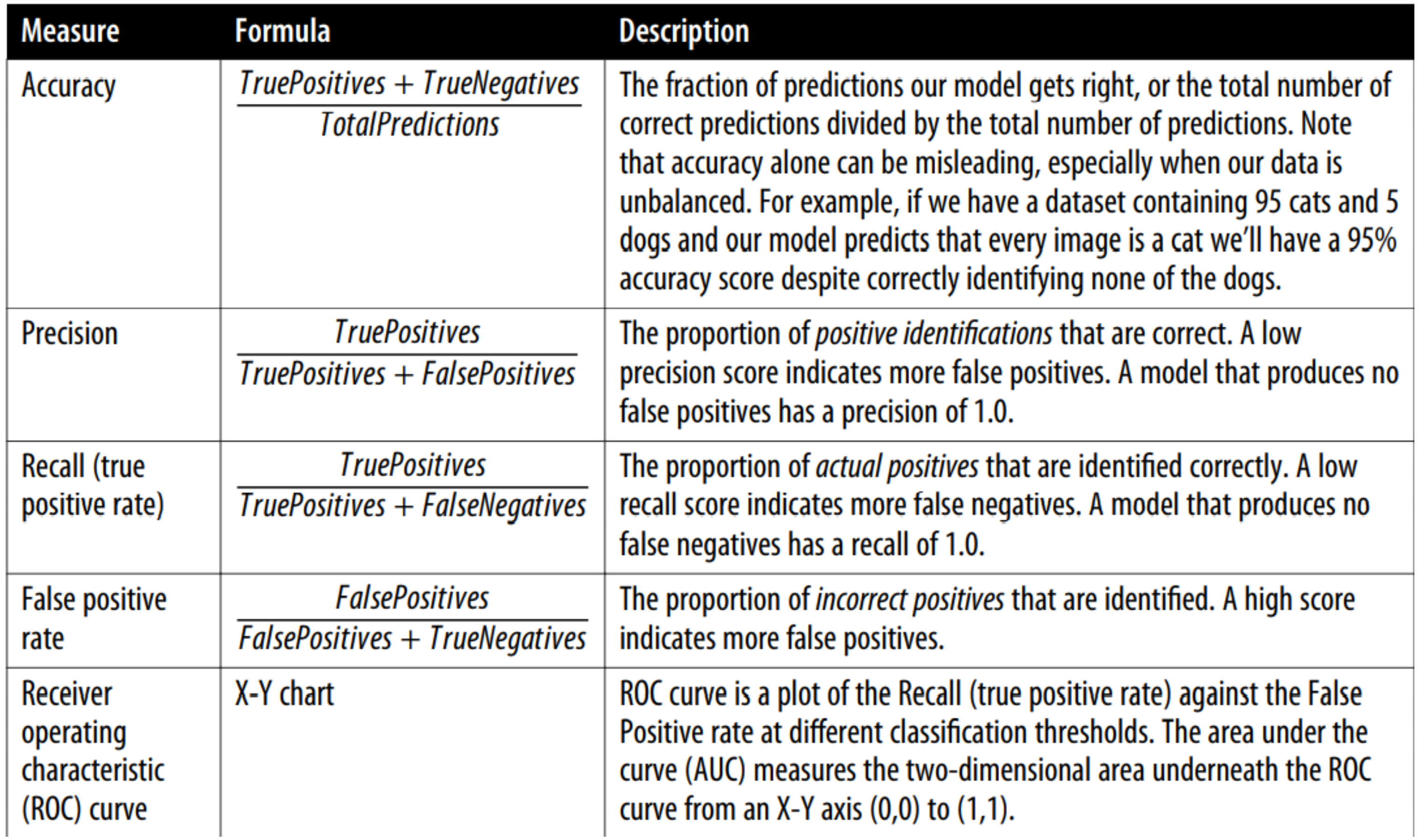
Now we want to train a model to predict missing links.



Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Predictive Metrics



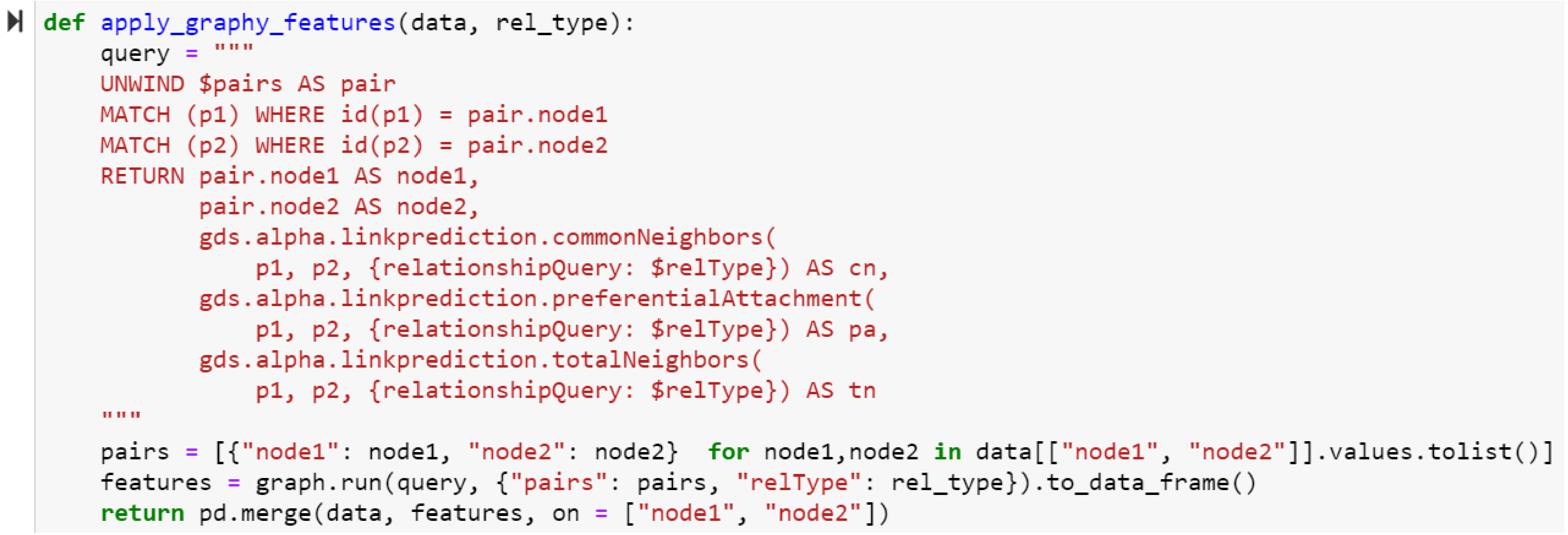
Measure	Formula	Description
Accuracy	$\frac{TruePositives + TrueNegatives}{TotalPredictions}$	The fraction of predictions our model gets right, or the total number of correct predictions divided by the total number of predictions. Note that accuracy alone can be misleading, especially when our data is unbalanced. For example, if we have a dataset containing 95 cats and 5 dogs and our model predicts that every image is a cat we'll have a 95% accuracy score despite correctly identifying none of the dogs.
Precision	$\frac{TruePositives}{TruePositives + FalsePositives}$	The proportion of *positive identifications* that are correct. A low precision score indicates more false positives. A model that produces no false positives has a precision of 1.0.
Recall (true positive rate)	$\frac{TruePositives}{TruePositives + FalseNegatives}$	The proportion of *actual positives* that are identified correctly. A low recall score indicates more false negatives. A model that produces no false negatives has a recall of 1.0.
False positive rate	$\frac{FalsePositives}{FalsePositives + TrueNegatives}$	The proportion of *incorrect positives* that are identified. A high score indicates more false positives.
Receiver operating characteristic (ROC) curve	X-Y chart	ROC curve is a plot of the Recall (true positive rate) against the False Positive rate at different classification thresholds. The area under the curve (AUC) measures the two-dimensional area underneath the ROC curve from an X-Y axis (0,0) to (1,1).

We'll use accuracy, precision, recall, and ROC curves to evaluate our models. Accuracy is a coarse measure, so we'll focus on increasing our overall precision and recall measures. We'll use the ROC curves to compare how individual features change predictive rates.

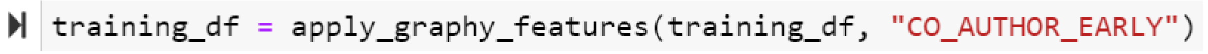
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

The function apply_graphy_features will compute and add the list of basic graph features to the training dataframe (common neighbours; preferential attachment; total neighbours).



```python
def apply_graphy_features(data, rel_type):
    query = """
    UNWIND $pairs AS pair
    MATCH (p1) WHERE id(p1) = pair.node1
    MATCH (p2) WHERE id(p2) = pair.node2
    RETURN pair.node1 AS node1,
           pair.node2 AS node2,
           gds.alpha.linkprediction.commonNeighbors(
               p1, p2, {relationshipQuery: $relType}) AS cn,
           gds.alpha.linkprediction.preferentialAttachment(
               p1, p2, {relationshipQuery: $relType}) AS pa,
           gds.alpha.linkprediction.totalNeighbors(
               p1, p2, {relationshipQuery: $relType}) AS tn
    """
    pairs = [{"node1": node1, "node2": node2}  for node1,node2 in data[["node1", "node2"]].values.tolist()]
    features = graph.run(query, {"pairs": pairs, "relType": rel_type}).to_data_frame()
    return pd.merge(data, features, on = ["node1", "node2"])
```

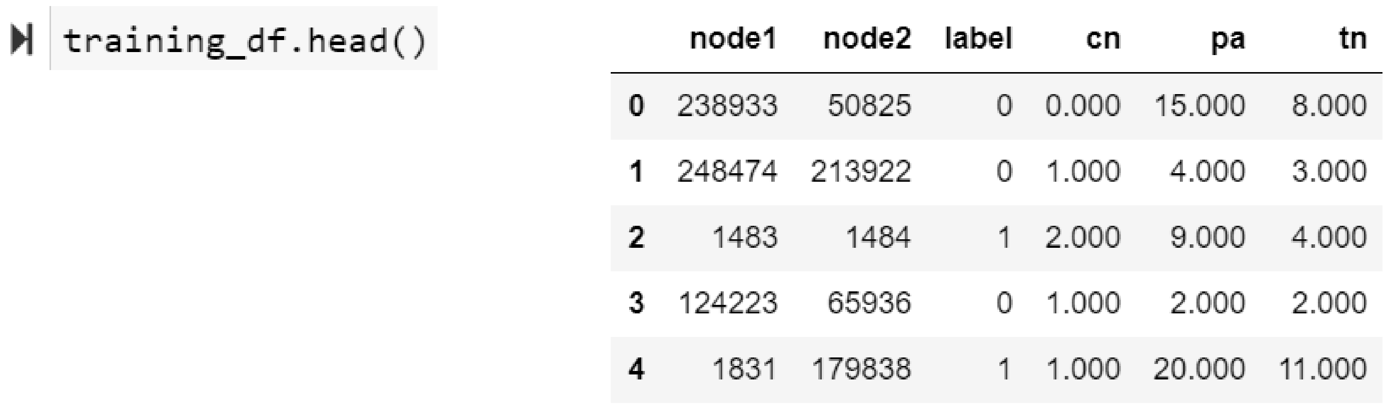


```python
training_df = apply_graphy_features(training_df, "CO_AUTHOR_EARLY")
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

The function apply_graphy_features will add the list of basic graph features to the training dataframes.



```
training_df.head()
```

	node1	node2	label	cn	pa	tn
0	238933	50825	0	0.000	15.000	8.000
1	248474	213922	0	1.000	4.000	3.000
2	1483	1484	1	2.000	9.000	4.000
3	124223	65936	0	1.000	2.000	2.000
4	1831	179838	1	1.000	20.000	11.000

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features

We do the same to the test dataframe.

```
test_df = apply_graphy_features(test_df, "CO_AUTHOR")
```

```
test_df.head()
```

	node1	node2	label	cn	pa	tn
0	10748	171011	1	1.000	12.000	7.000
1	101991	242673	1	2.000	42.000	15.000
2	247856	256368	1	1.000	8.000	5.000
3	156340	228229	1	3.000	64.000	17.000
4	255681	255685	1	8.000	81.000	10.000

Graph Algorithms and Machine Learning – Case Study

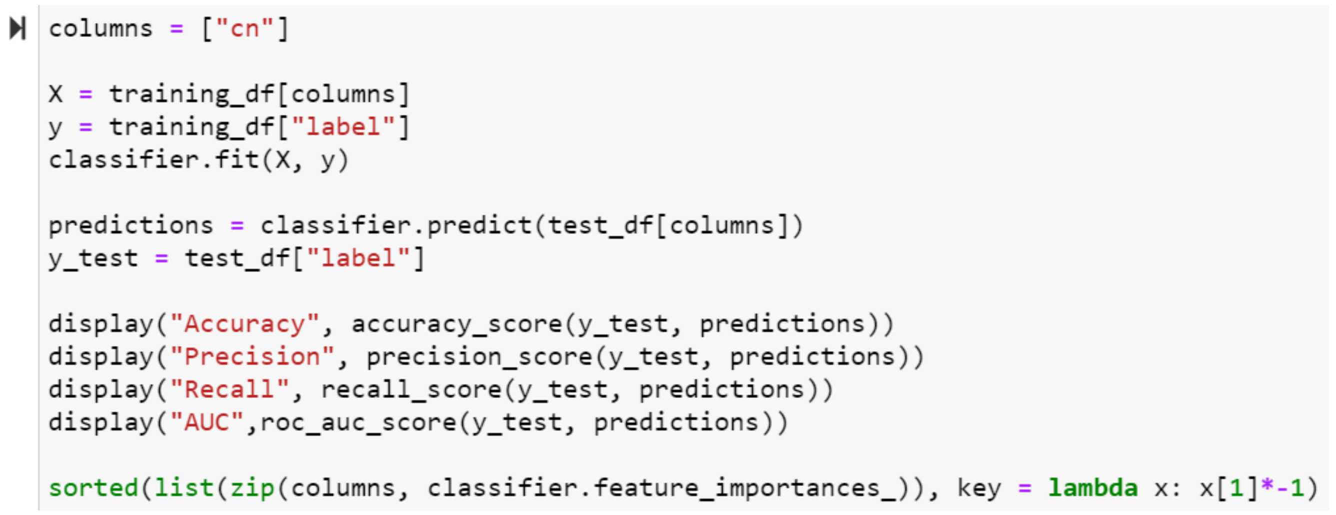
Citations Link Prediction – Basic Graph Features

We now build a model based on these graphy features. We start by just using one of the features - common neighbours.

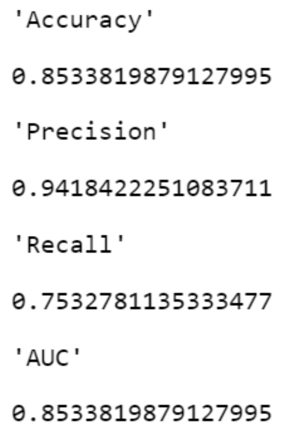
The following code builds a random forest model, evaluates it against the test dataset, and then indicates which of the features had the most importance in the model.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features - cn



```
columns = ["cn"]

X = training_df[columns]
y = training_df["label"]
classifier.fit(X, y)

predictions = classifier.predict(test_df[columns])
y_test = test_df["label"]

display("Accuracy", accuracy_score(y_test, predictions))
display("Precision", precision_score(y_test, predictions))
display("Recall", recall_score(y_test, predictions))
display("AUC",roc_auc_score(y_test, predictions))

sorted(list(zip(columns, classifier.feature_importances_)), key = lambda x: x[1]*-1)
```

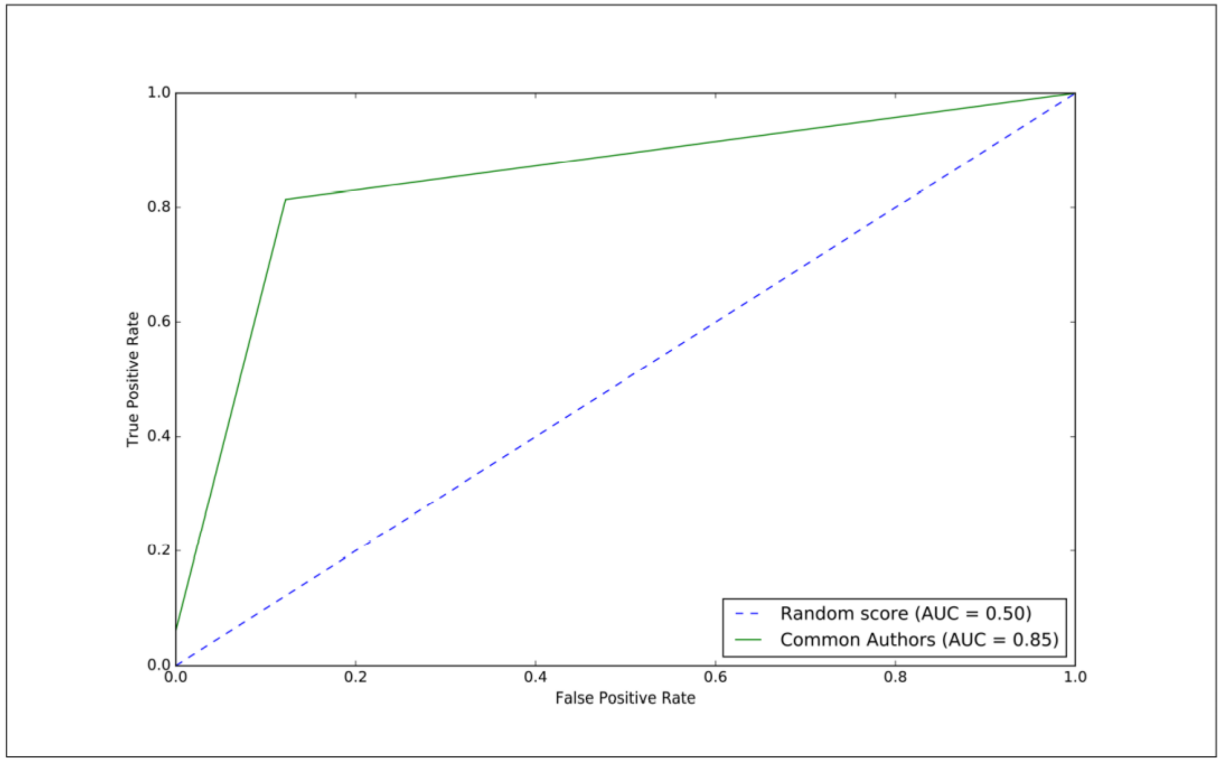


```
'Accuracy'

0.8533819879127995

'Precision'

0.9418422251083711

'Recall'

0.7532781135333477

'AUC'

0.8533819879127995
```

The scores for accuracy; precision and AUC are good, but recall isn't very good. The model has a precision of 0.941842, which means it’s very good at predicting that links exist. However, our recall is 0.753278, which means it’s not good at predicting when links do not exist. We can also plot the ROC curve (correlation of true positives and False positives).

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Basic Graph Features - cn



The ROC curve for basic model

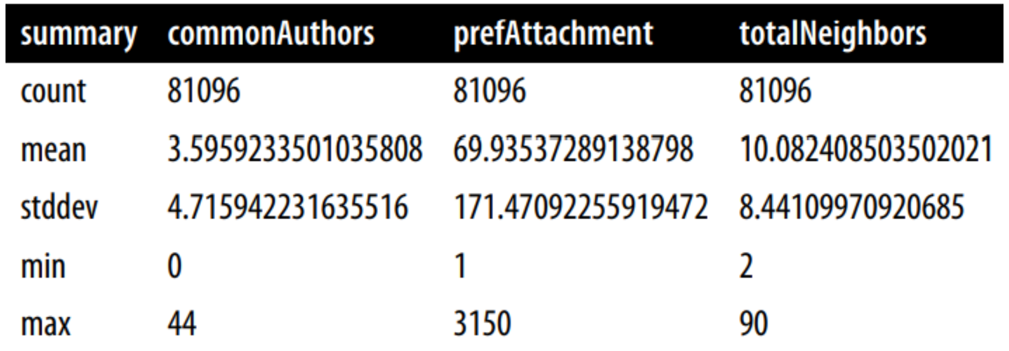
The common authors model gives us a 0.86 area under the curve (AUC) score. Although this gives us one overall predictive measure, we need the chart (or other measures) to evaluate whether this fits our goal. We see that as we get close to an 80% true positive rate (recall) our false positive rate reaches about 20%. That could be problematic in scenarios like fraud detection where false positives are expensive to chase.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Graphy Features – cn; pn; tn

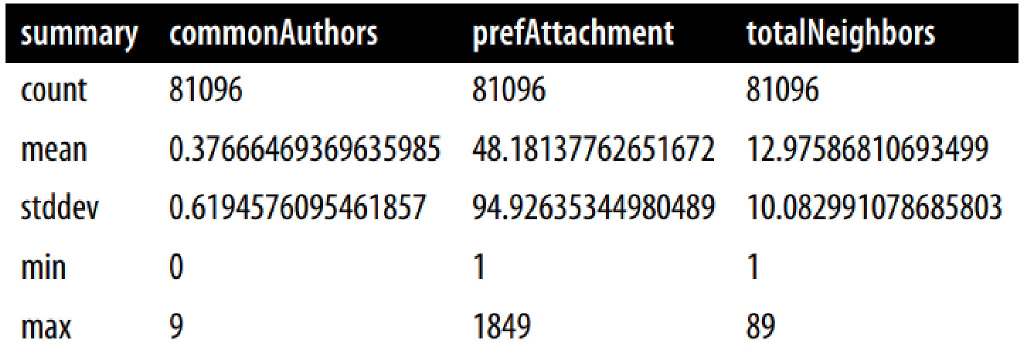
Before we train our model, let’s explore how the data is distributed.

Label=1



summary	commonAuthors	prefAttachment	totalNeighbors
count	81096	81096	81096
mean	3.5959233501035808	69.93537289138798	10.082408503502021
stddev	4.715942231635516	171.47092255919472	8.44109970920685
min	0	1	2
max	44	3150	90

Label=0



summary	commonAuthors	prefAttachment	totalNeighbors
count	81096	81096	81096
mean	0.37666469369635985	48.18137762651672	12.97586810693499
stddev	0.6194576095461857	94.92635344980489	10.082991078685803
min	0	1	1
max	9	1849	89

Graph Algorithms and Machine Learning – Case Study

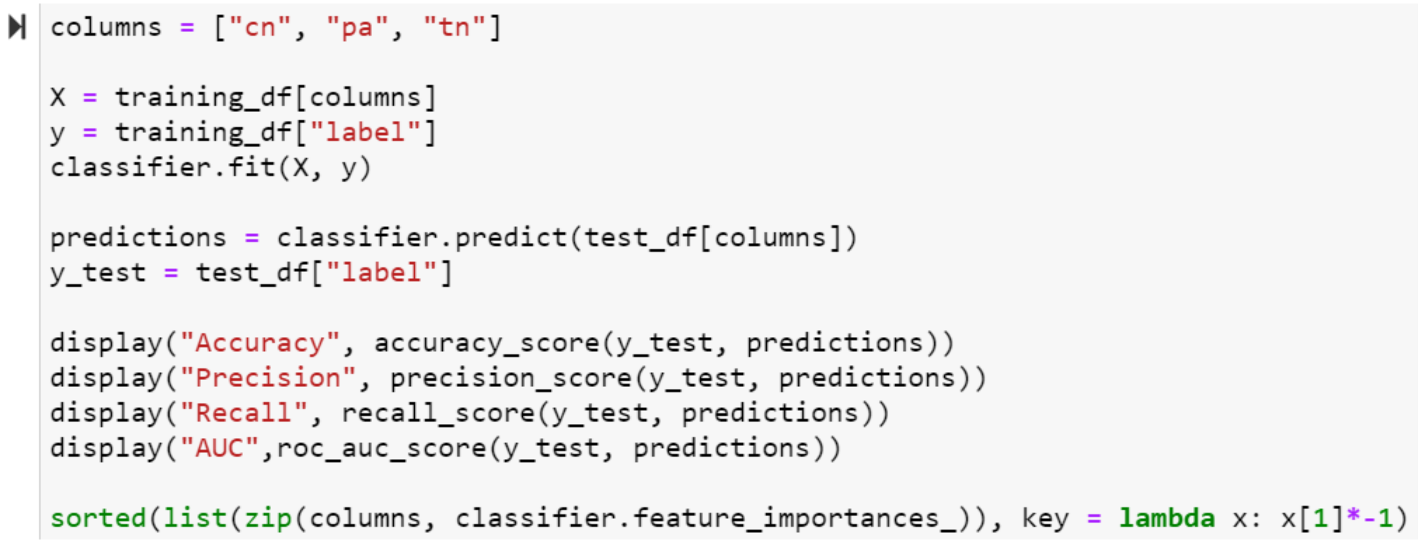
Citations Link Prediction – Graphy Features – cn; pn; tn

Features with larger differences between links (co-authorship) and no link (no co-authorship) should be more predictive because the divide is greater. The average value for prefAttachment is higher for authors who have collaborated versus those who haven’t. That difference is even more substantial for commonAuthors. We notice that there isn’t much difference in the values for totalNeighbors, which probably means this feature won’t be very predictive. Also interesting is the large standard deviation as well as the minimum and maximum values for preferential attachment. This is what we might expect for small-world networks with concentrated hubs (super-connectors).

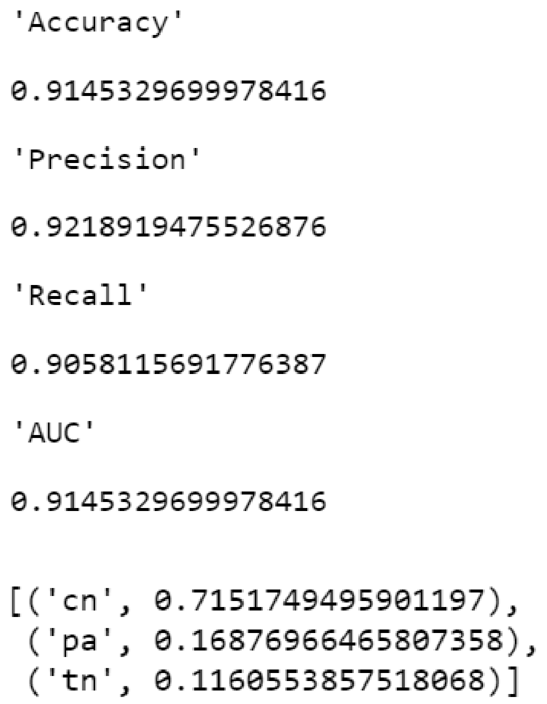
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Graphy Features – cn; pn; tn

Now we will train a new model, adding preferential attachment and total union of neighbours.



```
columns = ["cn", "pa", "tn"]

X = training_df[columns]
y = training_df["label"]
classifier.fit(X, y)

predictions = classifier.predict(test_df[columns])
y_test = test_df["label"]

display("Accuracy", accuracy_score(y_test, predictions))
display("Precision", precision_score(y_test, predictions))
display("Recall", recall_score(y_test, predictions))
display("AUC",roc_auc_score(y_test, predictions))

sorted(list(zip(columns, classifier.feature_importances_)), key = lambda x: x[1]*-1)
```



```
'Accuracy'

0.9145329699978416

'Precision'

0.9218919475526876

'Recall'

0.9058115691776387

'AUC'

0.9145329699978416


[('cn', 0.7151749495901197),
 ('pa', 0.16876966465807358),
 ('tn', 0.1160553857518068)]
```

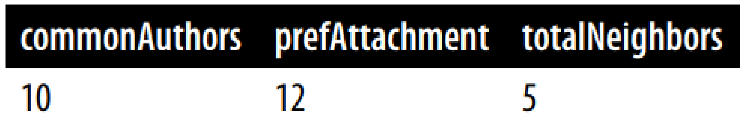
Accuracy and recall have increased substantially, but the precision has dropped a bit and we're still misclassifying about 8% of the links.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction

Of the three features we've used so far, commonAuthors is the most important feature by a large margin. To understand how our predictive models are created, we can visualize one of the decision trees in our random forest using a GraphViz file.

If we use this decision tree to predict whether a pair of nodes with the following features are linked.

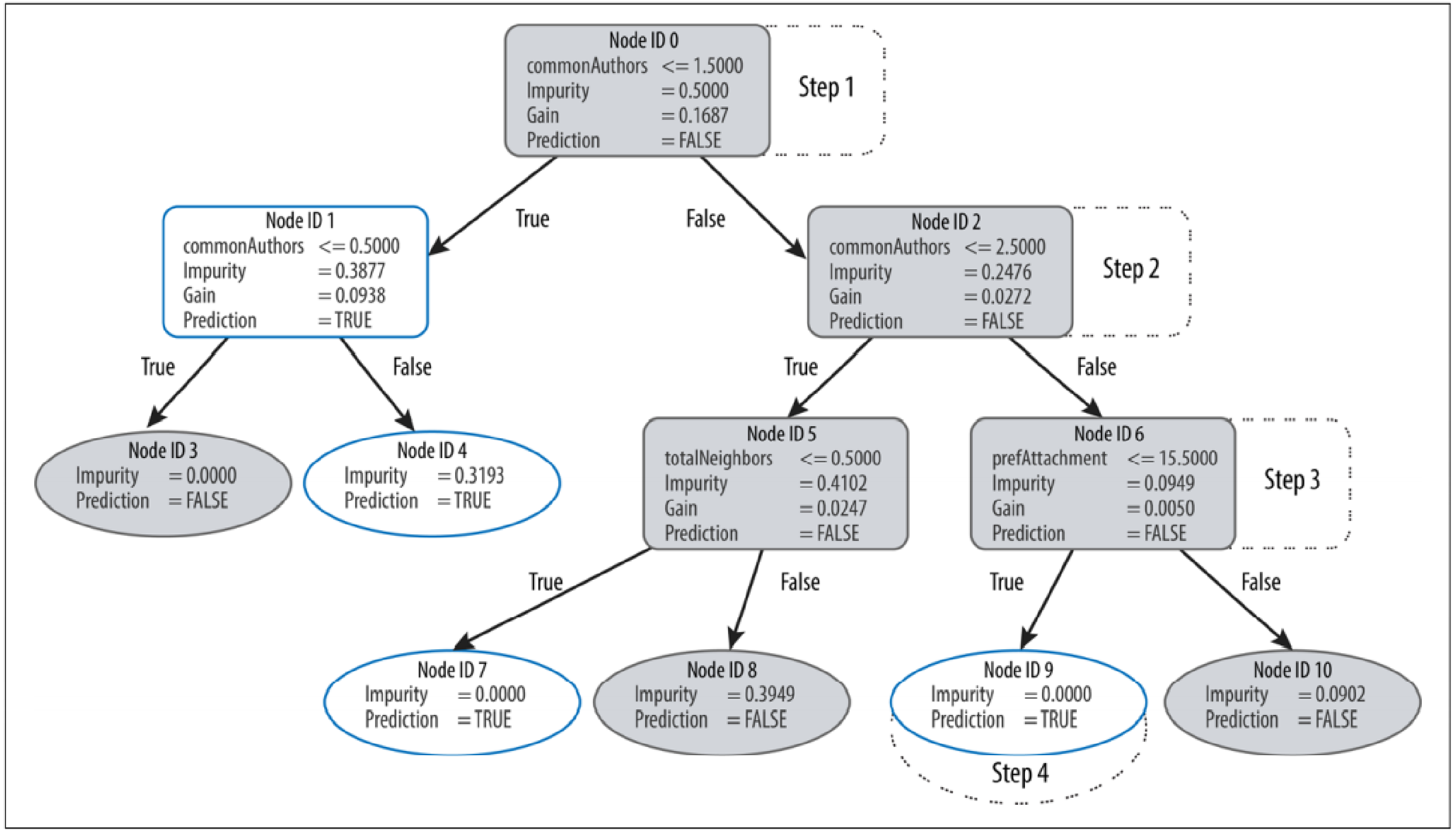


commonAuthors	prefAttachment	totalNeighbors
10	12	5

The random forest walks through several steps to create a prediction.

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction



Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction

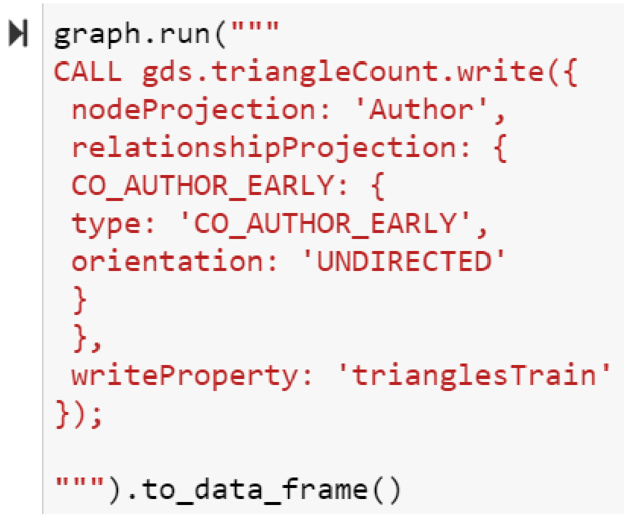
1. We start from node 0, where we have more than 1.5 commonAuthors, so we follow the False branch down to node 2.
2. We have more than 2.5 commonAuthors here, so we follow the False branch to node 6.
3. We have a score of less than 15.5 for prefAttachment, which takes us to node 9.
4. Node 9 is a leaf node in this decision tree, which means that we don't have to check any more conditions—the value of Prediction (i.e., True) on this node is the decision tree's prediction.
5. Finally, the random forest evaluates the item being predicted against a collection of these decision trees and makes its prediction based on the most popular out-come.

Now we will look at adding more graph features

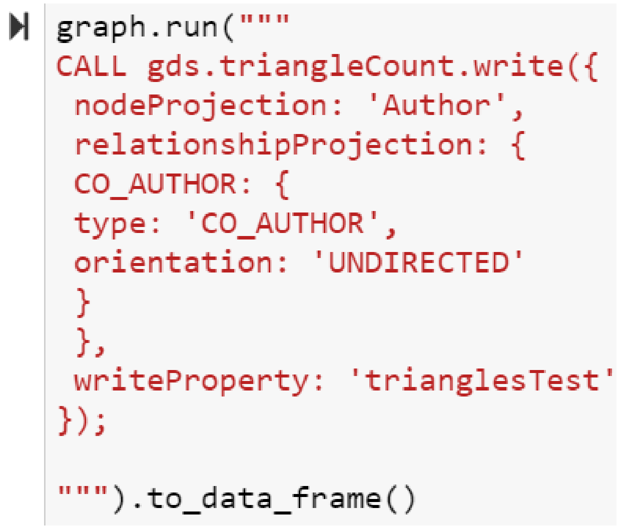
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Triangles & Clustering Coeff.

Recommendation solutions often base predictions on some form of triangle metric, so let's see if they further help with the case study. We can compute the number of triangles that a node is a part of and the clustering coefficient of the node by running the following queries.



```
graph.run("""
CALL gds.triangleCount.write({
 nodeProjection: 'Author',
 relationshipProjection: {
 CO_AUTHOR_EARLY: {
 type: 'CO_AUTHOR_EARLY',
 orientation: 'UNDIRECTED'
 }
 },
 writeProperty: 'trianglesTrain'
});

""").to_data_frame()
```

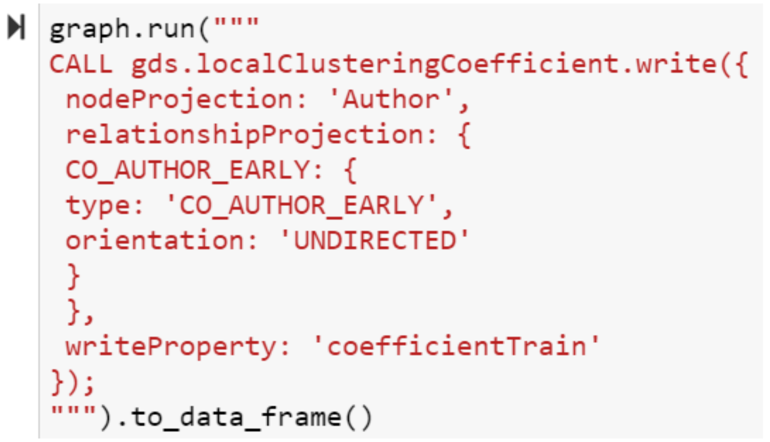


```
graph.run("""
CALL gds.triangleCount.write({
 nodeProjection: 'Author',
 relationshipProjection: {
 CO_AUTHOR: {
 type: 'CO_AUTHOR',
 orientation: 'UNDIRECTED'
 }
 },
 writeProperty: 'trianglesTest'
});

""").to_data_frame()
```

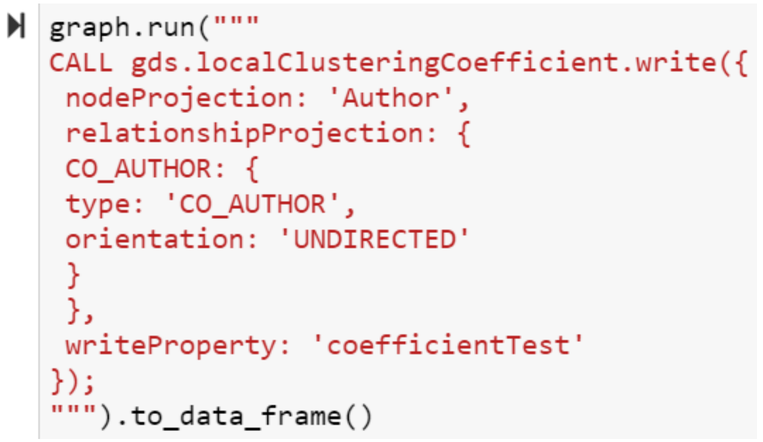
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Triangles & Clustering Coeff.

And its clustering coefficient by running the following query.



```
graph.run("""
CALL gds.localClusteringCoefficient.write({
 nodeProjection: 'Author',
 relationshipProjection: {
 CO_AUTHOR_EARLY: {
 type: 'CO_AUTHOR_EARLY',
 orientation: 'UNDIRECTED'
 }
 },
 writeProperty: 'coefficientTrain'
});
""").to_data_frame()
```

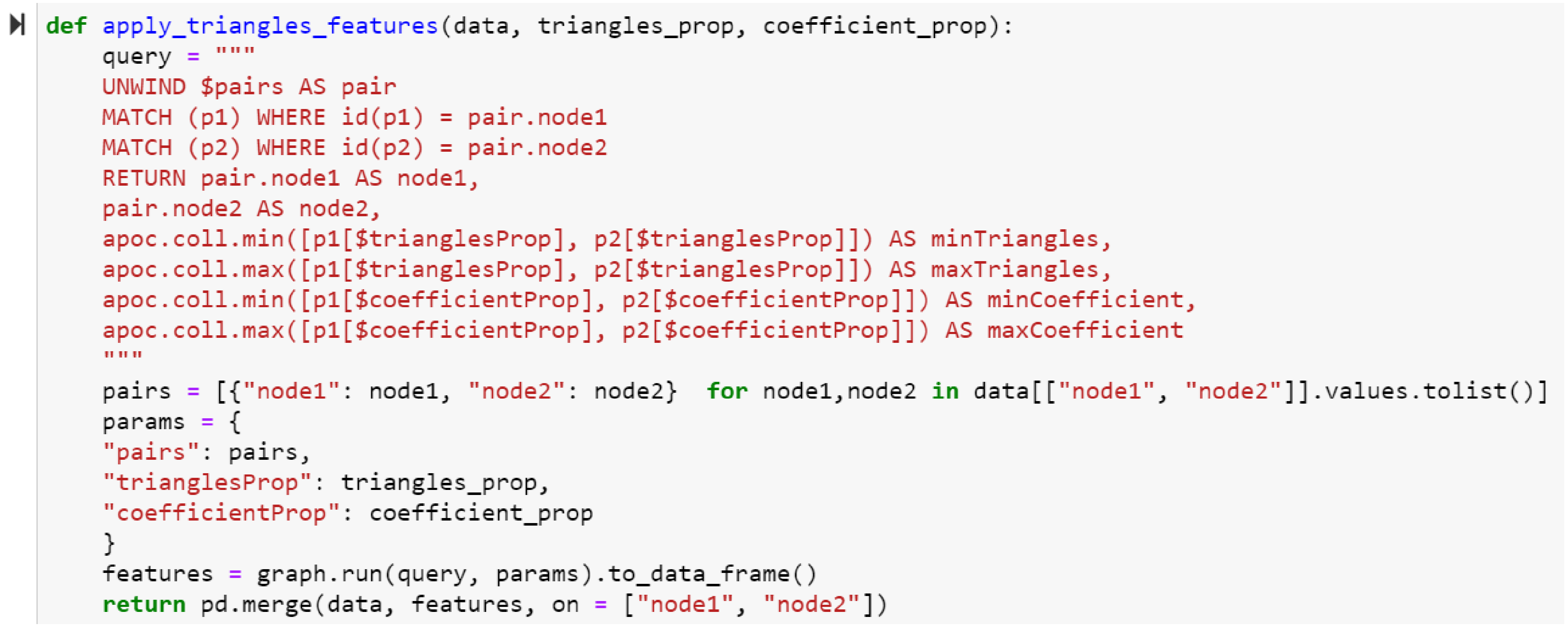


```
graph.run("""
CALL gds.localClusteringCoefficient.write({
 nodeProjection: 'Author',
 relationshipProjection: {
 CO_AUTHOR: {
 type: 'CO_AUTHOR',
 orientation: 'UNDIRECTED'
 }
 },
 writeProperty: 'coefficientTest'
});
""").to_data_frame()
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Triangles & Clustering Coeff.

The following function will add these features to the DataFrames:



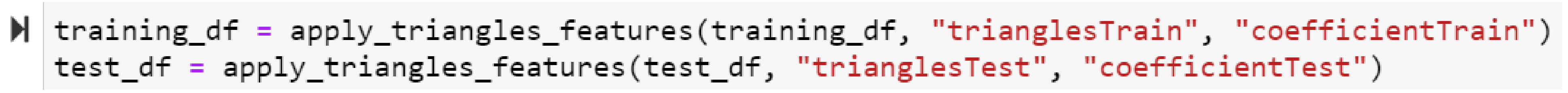
```
def apply_triangles_features(data, triangles_prop, coefficient_prop):
    query = """
    UNWIND $pairs AS pair
    MATCH (p1) WHERE id(p1) = pair.node1
    MATCH (p2) WHERE id(p2) = pair.node2
    RETURN pair.node1 AS node1,
    pair.node2 AS node2,
    apoc.coll.min([p1[$trianglesProp], p2[$trianglesProp]]) AS minTriangles,
    apoc.coll.max([p1[$trianglesProp], p2[$trianglesProp]]) AS maxTriangles,
    apoc.coll.min([p1[$coefficientProp], p2[$coefficientProp]]) AS minCoefficient,
    apoc.coll.max([p1[$coefficientProp], p2[$coefficientProp]]) AS maxCoefficient
    """
    pairs = [{"node1": node1, "node2": node2}  for node1,node2 in data[["node1", "node2"]].values.tolist()]
    params = {
    "pairs": pairs,
    "trianglesProp": triangles_prop,
    "coefficientProp": coefficient_prop
    }
    features = graph.run(query, params).to_data_frame()
    return pd.merge(data, features, on = ["node1", "node2"])
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Triangles & Clustering Coeff.

Notice that we've used min and max prefixes for our triangle count and clustering coefficient algorithms. We need a way to prevent our model from learning based on the order authors in pairs are passed in from our undirected graph. To do this, we've split these features by the authors with minimum and maximum counts.

We can apply this function to our training and test DataFrames with the following code.

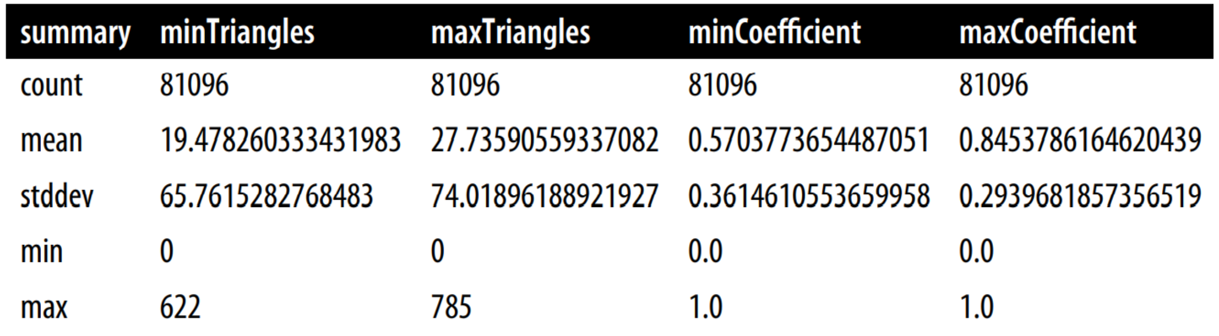


```
training_df = apply_triangles_features(training_df, "trianglesTrain", "coefficientTrain")
test_df = apply_triangles_features(test_df, "trianglesTest", "coefficientTest")
```

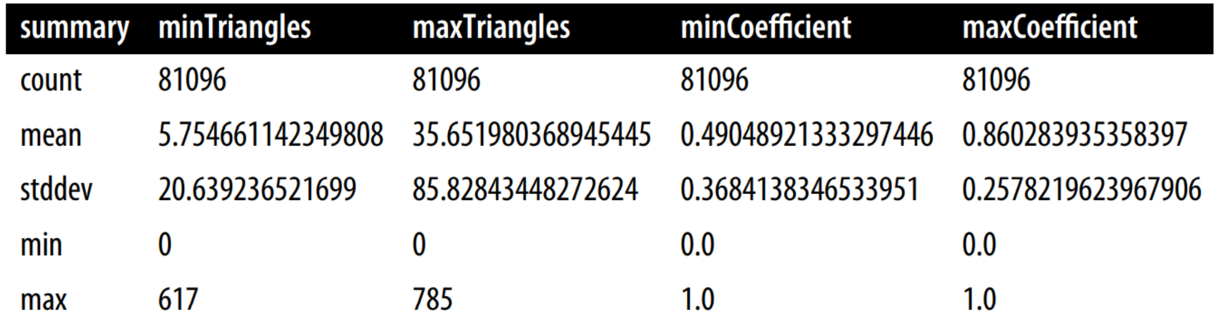
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Triangles & Clustering Coeff.

Descriptive statistics for each of our triangle features. Notice that there isn't a great difference between the co-authorship and no-co-authorship data. This could mean that these features aren't as predictive.



summary	minTriangles	maxTriangles	minCoefficient	maxCoefficient
count	81096	81096	81096	81096
mean	19.478260333431983	27.73590559337082	0.5703773654487051	0.8453786164620439
stddev	65.7615282768483	74.01896188921927	0.3614610553659958	0.2939681857356519
min	0	0	0.0	0.0
max	622	785	1.0	1.0

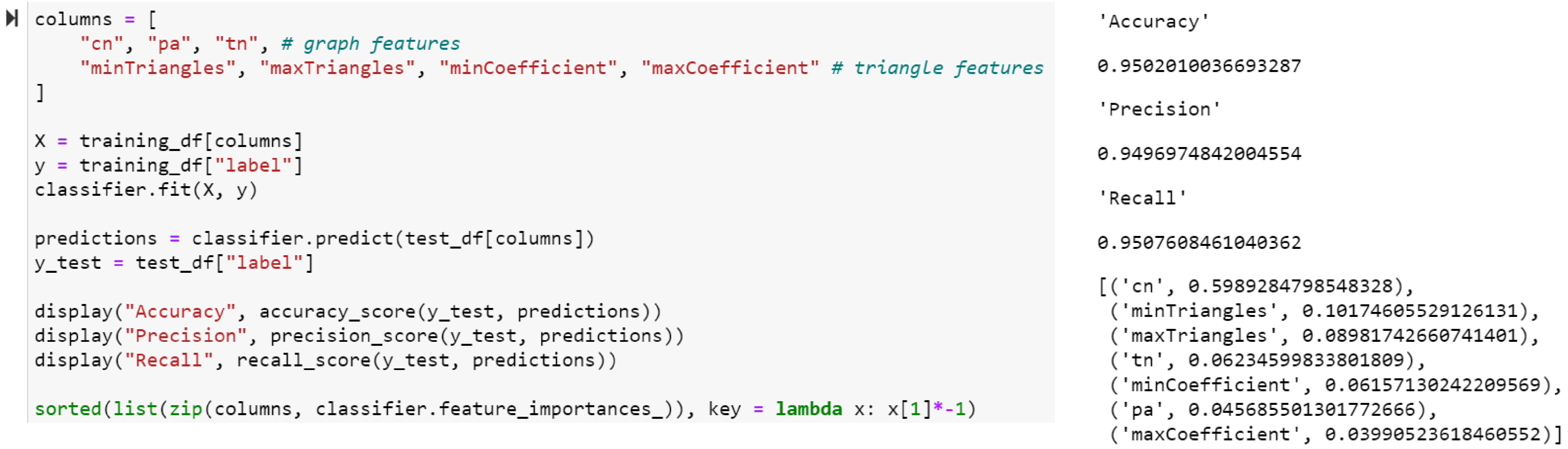


summary	minTriangles	maxTriangles	minCoefficient	maxCoefficient
count	81096	81096	81096	81096
mean	5.754661142349808	35.651980368945445	0.49048921333297446	0.860283935358397
stddev	20.639236521699	85.82843448272624	0.3684138346533951	0.2578219623967906
min	0	0	0.0	0.0
max	617	785	1.0	1.0

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Triangles & Clustering Coeff.

We can train another model by running the following code:

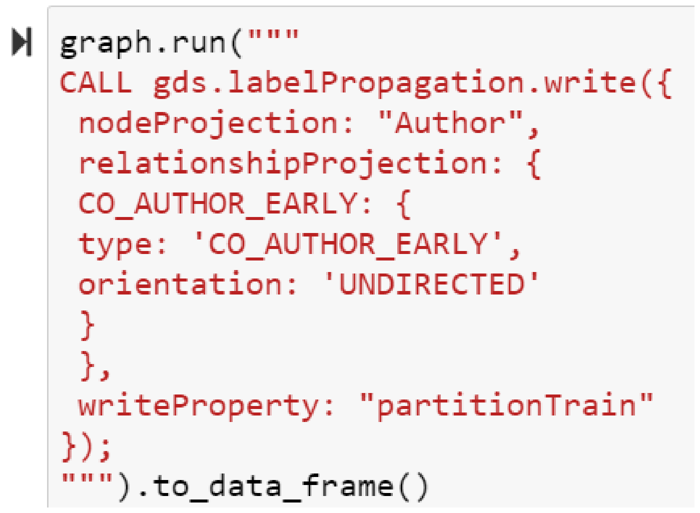


```
columns = [
    "cn", "pa", "tn", # graph features
    "minTriangles", "maxTriangles", "minCoefficient", "maxCoefficient" # triangle features
]

X = training_df[columns]
y = training_df["label"]
classifier.fit(X, y)

predictions = classifier.predict(test_df[columns])
y_test = test_df["label"]

display("Accuracy", accuracy_score(y_test, predictions))
display("Precision", precision_score(y_test, predictions))
display("Recall", recall_score(y_test, predictions))

sorted(list(zip(columns, classifier.feature_importances_)), key = lambda x: x[1]*-1)
```

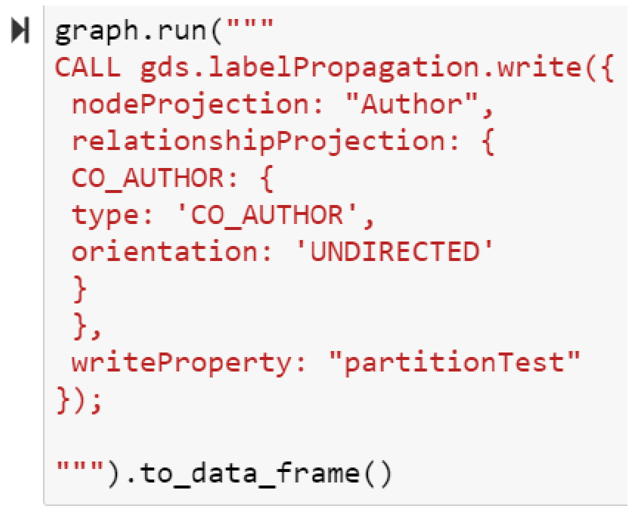
```
'Accuracy'

0.9502010036693287

'Precision'

0.9496974842004554

'Recall'

0.9507608461040362

[('cn', 0.5989284798548328),
 ('minTriangles', 0.10174605529126131),
 ('maxTriangles', 0.08981742660741401),
 ('tn', 0.06234599833801809),
 ('minCoefficient', 0.06157130242209569),
 ('pa', 0.04568550130177266),
 ('maxCoefficient', 0.03990523618460552)]
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Label Propagation



```
graph.run("""
CALL gds.labelPropagation.write({
 nodeProjection: "Author",
 relationshipProjection: {
 CO_AUTHOR_EARLY: {
 type: 'CO_AUTHOR_EARLY',
 orientation: 'UNDIRECTED'
 }
 },
 writeProperty: "partitionTrain"
});
""").to_data_frame()
```

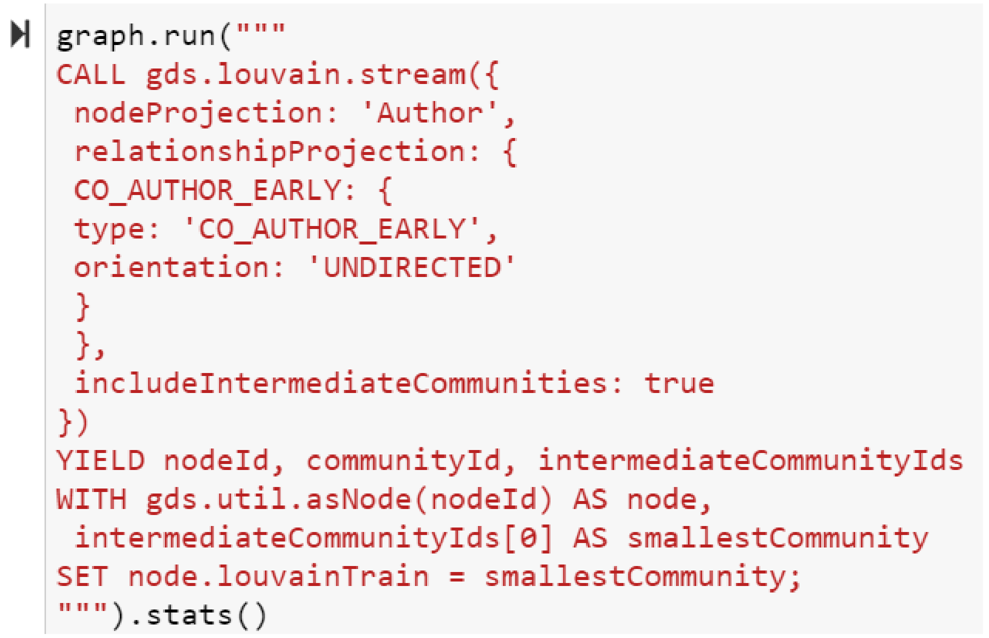


```
graph.run("""
CALL gds.labelPropagation.write({
 nodeProjection: "Author",
 relationshipProjection: {
 CO_AUTHOR: {
 type: 'CO_AUTHOR',
 orientation: 'UNDIRECTED'
 }
 },
 writeProperty: "partitionTest"
});

""").to_data_frame()
```

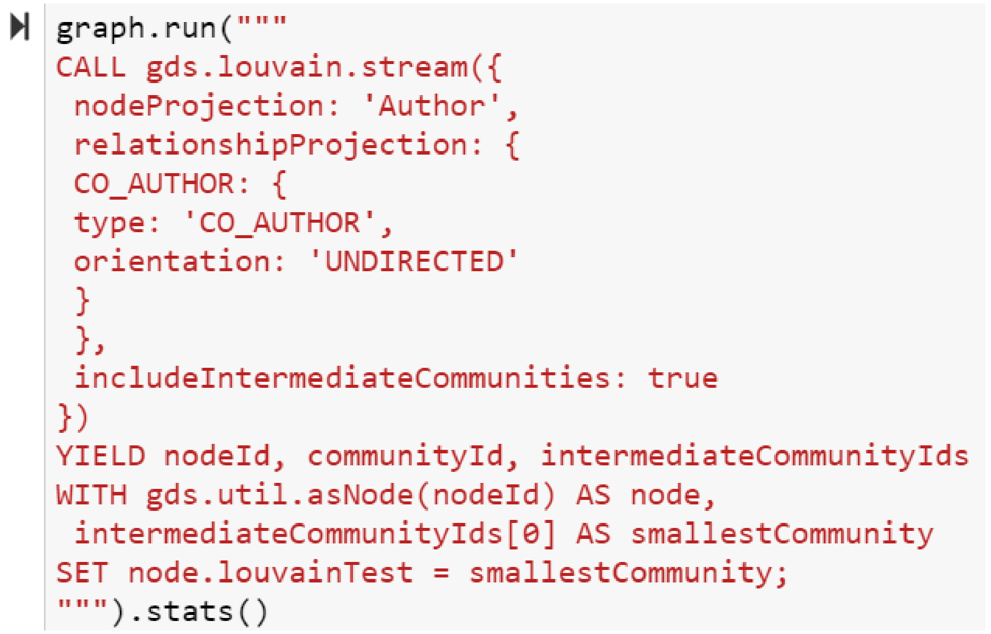
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Louvain

We'll also compute finer-grained groups using the Louvain algorithm. The Louvain algorithm returns intermediate clusters, and we'll store the smallest of these clusters in the property louvainTrain for the training set and louvainTest for the test set.



```
graph.run("""
CALL gds.louvain.stream({
 nodeProjection: 'Author',
 relationshipProjection: {
 CO_AUTHOR_EARLY: {
 type: 'CO_AUTHOR_EARLY',
 orientation: 'UNDIRECTED'
 }
 },
 includeIntermediateCommunities: true
})
YIELD nodeId, communityId, intermediateCommunityIds
WITH gds.util.asNode(nodeId) AS node,
 intermediateCommunityIds[0] AS smallestCommunity
SET node.louvainTrain = smallestCommunity;
""").stats()
```

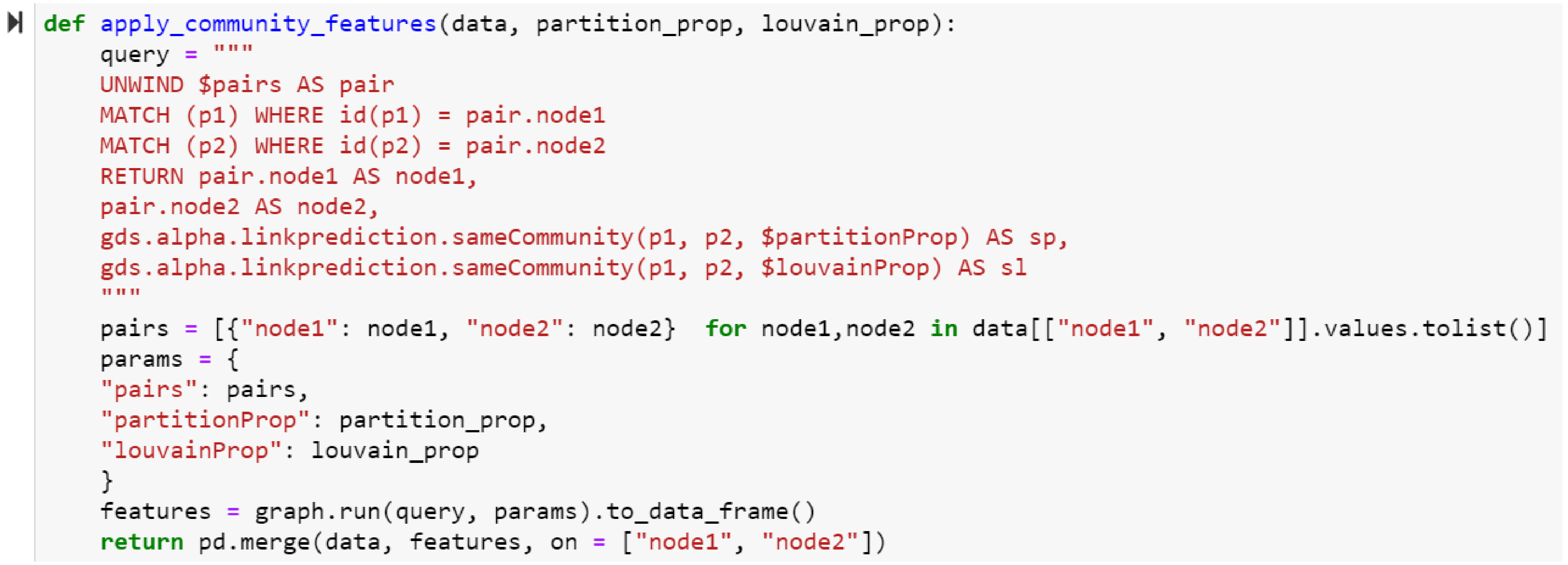


```
graph.run("""
CALL gds.louvain.stream({
 nodeProjection: 'Author',
 relationshipProjection: {
 CO_AUTHOR: {
 type: 'CO_AUTHOR',
 orientation: 'UNDIRECTED'
 }
 },
 includeIntermediateCommunities: true
})
YIELD nodeId, communityId, intermediateCommunityIds
WITH gds.util.asNode(nodeId) AS node,
 intermediateCommunityIds[0] AS smallestCommunity
SET node.louvainTest = smallestCommunity;
""").stats()
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Louvain

We'll now create the following function to return the values from these algorithms.

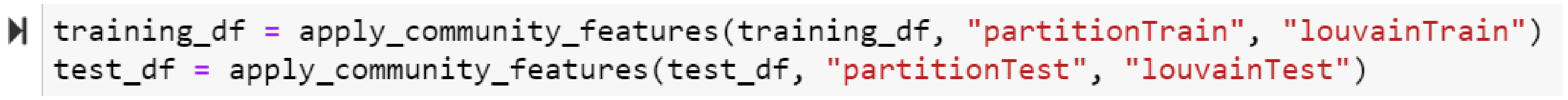


```
def apply_community_features(data, partition_prop, louvain_prop):
    query = """
    UNWIND $pairs AS pair
    MATCH (p1) WHERE id(p1) = pair.node1
    MATCH (p2) WHERE id(p2) = pair.node2
    RETURN pair.node1 AS node1,
    pair.node2 AS node2,
    gds.alpha.linkprediction.sameCommunity(p1, p2, $partitionProp) AS sp,
    gds.alpha.linkprediction.sameCommunity(p1, p2, $louvainProp) AS sl
    """
    pairs = [{"node1": node1, "node2": node2}  for node1,node2 in data[["node1", "node2"]].values.tolist()]
    params = {
    "pairs": pairs,
    "partitionProp": partition_prop,
    "louvainProp": louvain_prop
    }
    features = graph.run(query, params).to_data_frame()
    return pd.merge(data, features, on = ["node1", "node2"])
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Louvain

We can apply this function to our training and test DataFrames with the following code.

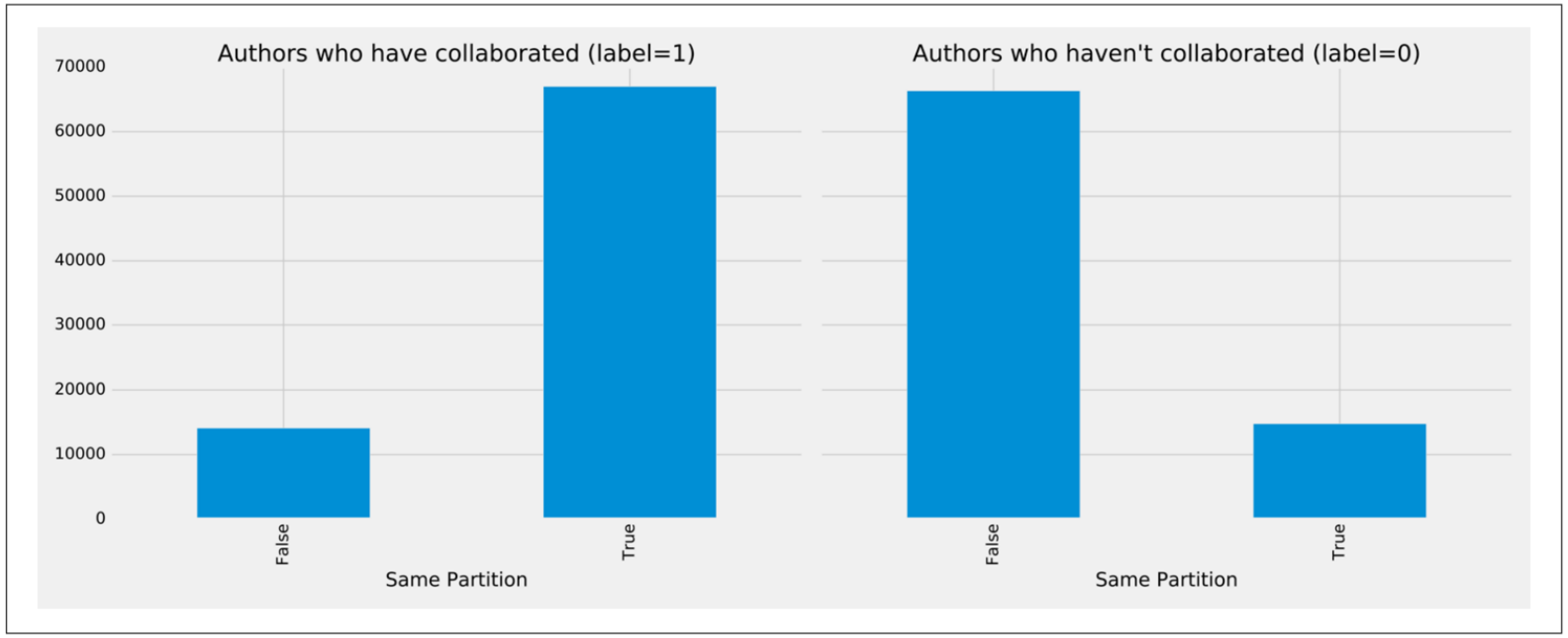


```
training_df = apply_community_features(training_df, "partitionTrain", "louvainTrain")
test_df = apply_community_features(test_df, "partitionTest", "louvainTest")
```

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Same Partition

Exploring whether pairs of nodes belong in the same partition. It looks like this feature could be quite predictive—authors who have collaborated are much more likely to be in the same partition than those who haven't.

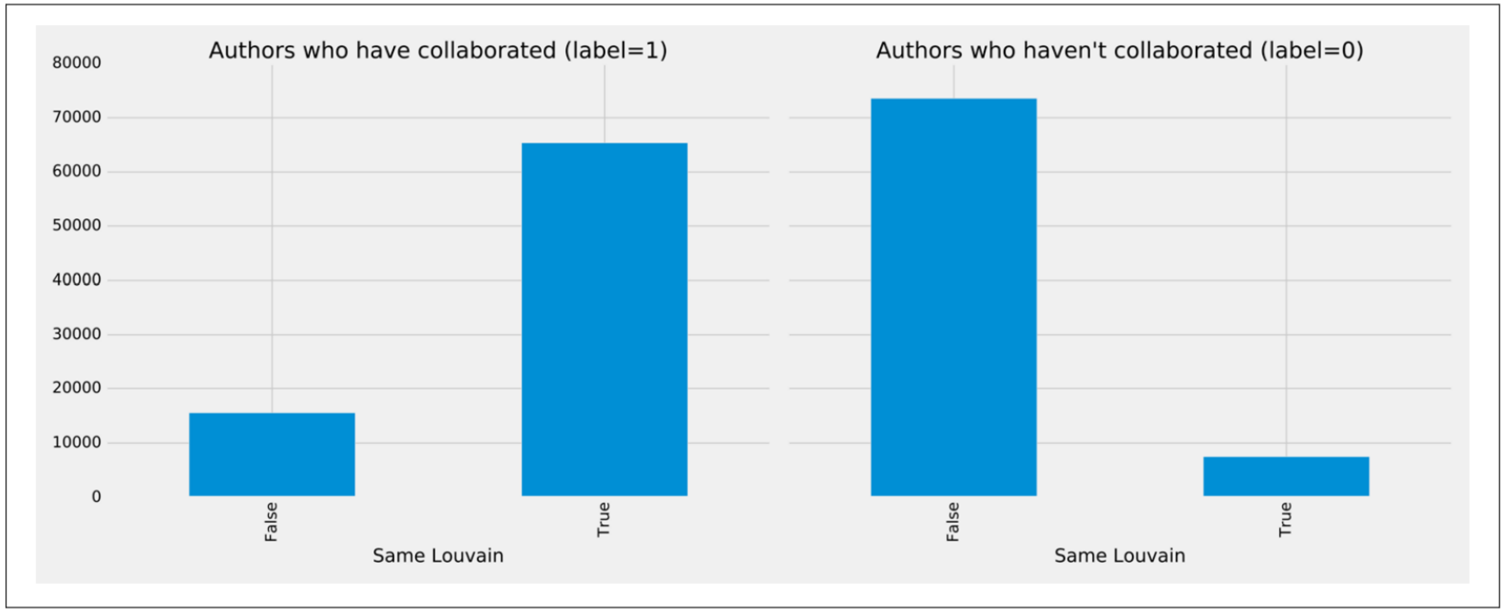


Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – Louvain

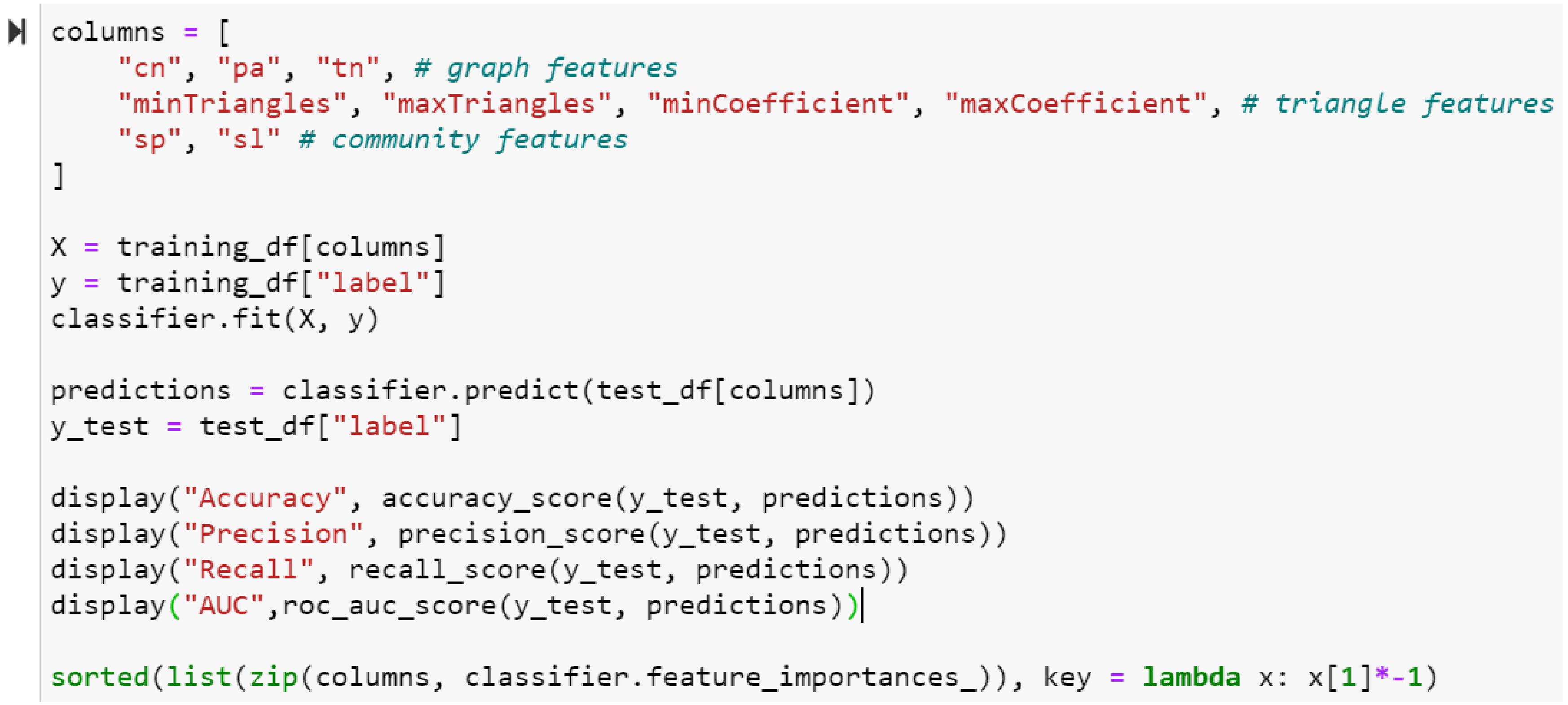
It looks like this Louvain could be quite predictive as well—authors who have collaborated are likely to be in the same cluster, and those who haven't are very unlikely to be in the same cluster.



Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction

We can train another model by running the following code.



```
columns = [
    "cn", "pa", "tn", # graph features
    "minTriangles", "maxTriangles", "minCoefficient", "maxCoefficient", # triangle features
    "sp", "sl" # community features
]

X = training_df[columns]
y = training_df["label"]
classifier.fit(X, y)

predictions = classifier.predict(test_df[columns])
y_test = test_df["label"]

display("Accuracy", accuracy_score(y_test, predictions))
display("Precision", precision_score(y_test, predictions))
display("Recall", recall_score(y_test, predictions))
display("AUC",roc_auc_score(y_test, predictions))

sorted(list(zip(columns, classifier.feature_importances_)), key = lambda x: x[1]*-1)
```

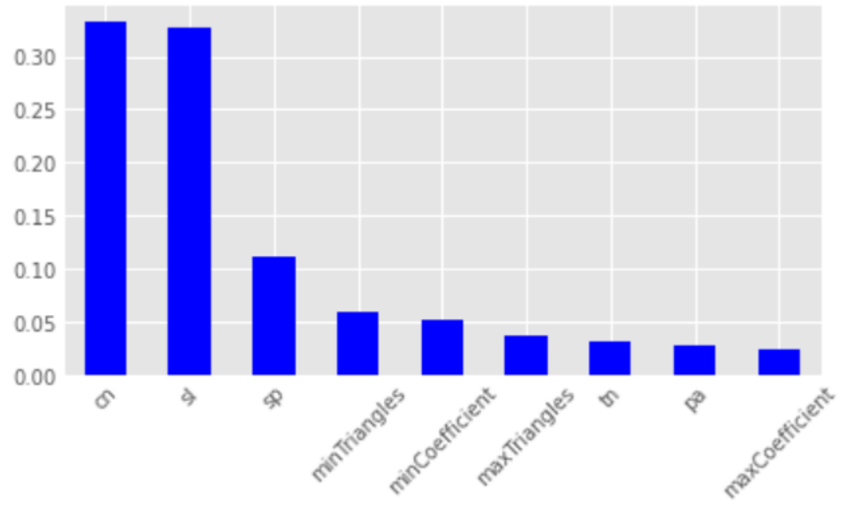
Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – All Features Engineered

```
'Accuracy'

0.9672728253831211

'Precision'

0.9626913520878416

'Recall'

0.9722237211310166

'AUC'

0.9672728253831211

[('cn', 0.3320235802034582),
 ('sl', 0.326798307400617),
 ('sp', 0.11175194242853856),
 ('minTriangles', 0.05906600652007075),
 ('minCoefficient', 0.05108531253088957),
 ('maxTriangles', 0.03658322901223461),
 ('tn', 0.031504739680862176),
 ('pa', 0.027726655588848646),
 ('maxCoefficient', 0.023460226634480576)]
```

```python
def plot_feature_importance(fields, feature_importances):
  df = pd.DataFrame({"Feature": fields, "Importance": feature_importances})
  df = df.sort_values("Importance", ascending=False)
  ax = df.plot(kind='bar', x='Feature', y='Importance', legend=None, color="blue")
  ax.xaxis.set_label_text("")
  plt.tight_layout()
  plt.xticks(rotation=45)
  plt.show()
```

```python
plot_feature_importance(columns, classifier.feature_importances_)
```

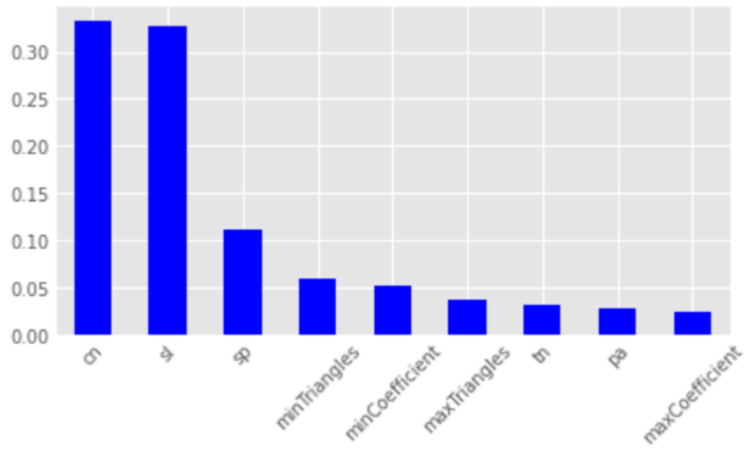


Graph Algorithms Mark Needham and Amy E. Hodler, O'Reilly 2019

Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – All Features Engineered

	graphy (cn only)	graphy (cn/pn/tn)	graphy triangle	graphy triangle community
Accuracy	85	91	95	**96**
Precision	94	92	94	**96**
Recall	75	90	95	**97**
AUC	85	91	95	**96**



Graph Algorithms and Machine Learning – Case Study

Citations Link Prediction – All Features Engineered

Although the common authors model is overall very important, it’s good to avoid having an overly dominant element that might skew predictions on new data. Community detection algorithms had a lot of influence in our last model with all the features included, and this helps round out our predictive approach.

We now have a good, balanced model for predicting co-authorship links. Using graphs for connected feature extraction can significantly improve our predictions. The ideal graph features and algorithms vary depending on the attributes of the data, including the network domain and graph shape.