

Node Embeddings

This notebook demonstrates different methods for node embeddings and how to further reduce their dimensionality to be able to visualize them in a 2D plot.

Node embeddings are essentially an array of floating point numbers (length = embedding dimension) that can be used as "features" in machine learning. These numbers approximate the relationship and similarity information of each node and can also be seen as a way to encode the topology of the graph.

Considerations

Due to dimensionality reduction some information gets lost, especially when visualizing node embeddings in two dimensions. Nevertheless, it helps to get an intuition on what node embeddings are and how much of the similarity and neighborhood information is retained. The latter can be observed by how well nodes of the same color and therefore same community are placed together and how much bigger nodes with a high centrality score influence them.

If the visualization doesn't show a somehow clear separation between the communities (colors) here are some ideas for tuning:

- Clean the data, e.g. filter out very few nodes with extremely high degree that aren't actually that important
- Try directed vs. undirected projections
- Tune the embedding algorithm, e.g. use a higher dimensionality
- Tune t-SNE that is used to reduce the node embeddings dimension to two dimensions for visualization.

It could also be the case that the node embeddings are good enough and well suited the way they are despite their visualization for the down stream task like node classification or link prediction. In that case it makes sense to see how the whole pipeline performs before tuning the node embeddings in detail.

Note about data dependencies

PageRank centrality and Leiden community are also fetched from the Graph and need to be calculated first. This makes it easier to see if the embeddings approximate the structural information of the graph in the plot. If these properties are missing you will only see black dots all of the same size.

References

- [jqassistant](#)

- [Neo4j Python Driver](#)
- [Tutorial: Applied Graph Embeddings](#)
- [Visualizing the embeddings in 2D](#)
- [scikit-learn TSNE](#)
- [AttributeError: 'list' object has no attribute 'shape'](#)
- [Fast Random Projection \(neo4j\)](#)
- [HashGNN \(neo4j\)](#)
- [node2vec \(neo4j\)](#) computes a vector representation of a node based on second order random walks in the graph.
- [Complete guide to understanding Node2Vec algorithm](#)

The openTSNE version is: 1.0.1
The pandas version is: 1.5.1

Dimensionality reduction with t-distributed stochastic neighbor embedding (t-SNE)

The following function takes the original node embeddings with a higher dimensionality, e.g. 64 floating point numbers, and reduces them into a two dimensional array for visualization.

It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

(see <https://opentsne.readthedocs.io>)

1. Java Packages

1.1 Generate Node Embeddings using Fast Random Projection (Fast RP) for Java Packages

[Fast Random Projection](#) is used to reduce the dimensionality of the node feature space while preserving most of the distance information. Nodes with similar neighborhood result in node embedding with similar vectors.

👉 **Hint:** To skip existing node embeddings and always calculate them based on the parameters below edit `Node_EMBEDDINGS_0a_Query_Calculated` so that it won't return any results.

The results have been provided by the query filename: `../cypher/Node_EMBEDDINGS/Node_EMBEDDINGS_0a_Query_Calculated.cypher`

| | codeUnitName | shortCodeUnitName | projectName | communityId | centrality | embedding |
|---|----------------------------------|-------------------|-----------------|-------------|------------|---|
| 0 | org.axonframework.test | test | axon-test-4.9.3 | 0 | 0.081762 | [-0.22466301918029785, 0.25969457626342773, 0.... |
| 1 | org.axonframework.test.aggregate | aggregate | axon-test-4.9.3 | 0 | 0.016509 | [-0.26439180970191956, 0.21783214807510376, 0.... |
| 2 | org.axonframework.test.matchers | matchers | axon-test-4.9.3 | 0 | 0.033417 | [-0.21339406073093414, 0.24127791821956635, 0.... |
| 3 | org.axonframework.test.saga | saga | axon-test-4.9.3 | 0 | 0.016509 | [-0.21123544871807098, 0.22235634922981262, 0.... |
| 4 | org.axonframework.test.utils | utils | axon-test-4.9.3 | 0 | 0.017834 | [-0.140242338180542, 0.19431671500205994, 0.28... |

1.2 Dimensionality reduction with t-distributed stochastic neighbor embedding (t-SNE)

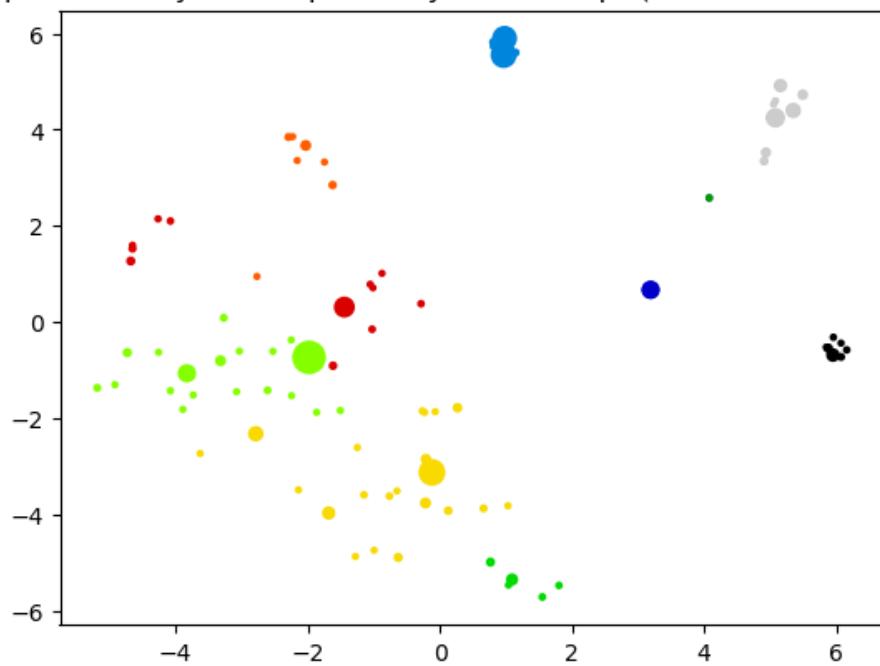
This step takes the original node embeddings with a higher dimensionality, e.g. 64 floating point numbers, and reduces them into a two dimensional array for visualization. For more details look up the function declaration for "prepare_node_embeddings_for_2d_visualization".

```
-----  
TSNE(early_exaggeration=12, random_state=47, verbose=1)  
-----  
====> Finding 90 nearest neighbors using exact search using euclidean distance...  
    --> Time elapsed: 0.02 seconds  
====> Calculating affinity matrix...  
    --> Time elapsed: 0.00 seconds  
====> Calculating PCA-based initialization...  
    --> Time elapsed: 0.00 seconds  
====> Running optimization with exaggeration=12.00, lr=7.75 for 250 iterations...  
Iteration  50, KL divergence -1.2221, 50 iterations in 0.0359 sec  
Iteration 100, KL divergence 0.9941, 50 iterations in 0.0136 sec  
Iteration 150, KL divergence 0.9941, 50 iterations in 0.0123 sec  
Iteration 200, KL divergence 0.9941, 50 iterations in 0.0123 sec  
Iteration 250, KL divergence 0.9941, 50 iterations in 0.0122 sec  
    --> Time elapsed: 0.09 seconds  
====> Running optimization with exaggeration=1.00, lr=93.00 for 500 iterations...  
Iteration  50, KL divergence 0.1074, 50 iterations in 0.0367 sec  
Iteration 100, KL divergence 0.0793, 50 iterations in 0.0492 sec  
Iteration 150, KL divergence 0.0739, 50 iterations in 0.0343 sec  
Iteration 200, KL divergence 0.0736, 50 iterations in 0.0332 sec  
Iteration 250, KL divergence 0.0758, 50 iterations in 0.0328 sec  
Iteration 300, KL divergence 0.0762, 50 iterations in 0.0326 sec  
Iteration 350, KL divergence 0.0763, 50 iterations in 0.0329 sec  
Iteration 400, KL divergence 0.0764, 50 iterations in 0.0330 sec  
Iteration 450, KL divergence 0.0764, 50 iterations in 0.0333 sec  
Iteration 500, KL divergence 0.0764, 50 iterations in 0.0330 sec  
    --> Time elapsed: 0.35 seconds  
(93, 2)
```

| | codeUnit | artifact | communityId | centrality | x | y |
|---|----------------------------------|-----------------|-------------|------------|----------|-----------|
| 0 | org.axonframework.test | axon-test-4.9.3 | 0 | 0.081762 | 5.952629 | -0.686031 |
| 1 | org.axonframework.test.aggregate | axon-test-4.9.3 | 0 | 0.016509 | 5.849969 | -0.522497 |
| 2 | org.axonframework.test.matchers | axon-test-4.9.3 | 0 | 0.033417 | 5.881065 | -0.543739 |
| 3 | org.axonframework.test.saga | axon-test-4.9.3 | 0 | 0.016509 | 6.074457 | -0.437148 |
| 4 | org.axonframework.test.utils | axon-test-4.9.3 | 0 | 0.017834 | 5.958408 | -0.315026 |

1.3 Visualization of the node embeddings reduced to two dimensions

Java Package positioned by their dependency relationships (FastRP node embeddings + t-SNE)



1.4 Node Embeddings for Java Packages using HashGNN

HashGNN resembles Graph Neural Networks (GNN) but does not include a model or require training. It combines ideas of GNNs and fast randomized algorithms. For more details see [HashGNN](#). Here, the latter 3 steps are combined into one for HashGNN.

The results have been provided by the query filename: `.../cypher/Node_EMBEDDINGS/Node_EMBEDDINGS_0a_Query_Calculated.cypher`

| | codeUnitName | shortCodeUnitName | projectName | communityId | centrality | embedding |
|---|----------------------------------|-------------------|-----------------|-------------|------------|---|
| 0 | org.axonframework.test | test | axon-test-4.9.3 | 0 | 0.081762 | [0.4330126941204071, -1.5155444294214249, 0.21... |
| 1 | org.axonframework.test.aggregate | aggregate | axon-test-4.9.3 | 0 | 0.016509 | [0.21650634706020355, -1.2990380823612213, 0.2... |
| 2 | org.axonframework.test.matchers | matchers | axon-test-4.9.3 | 0 | 0.033417 | [0.4330126941204071, -1.5155444294214249, 0.21... |
| 3 | org.axonframework.test.saga | saga | axon-test-4.9.3 | 0 | 0.016509 | [0.4330126941204071, -1.5155444294214249, 0.21... |
| 4 | org.axonframework.test.utils | utils | axon-test-4.9.3 | 0 | 0.017834 | [0.4330126941204071, -1.2990380823612213, 0.43... |

```

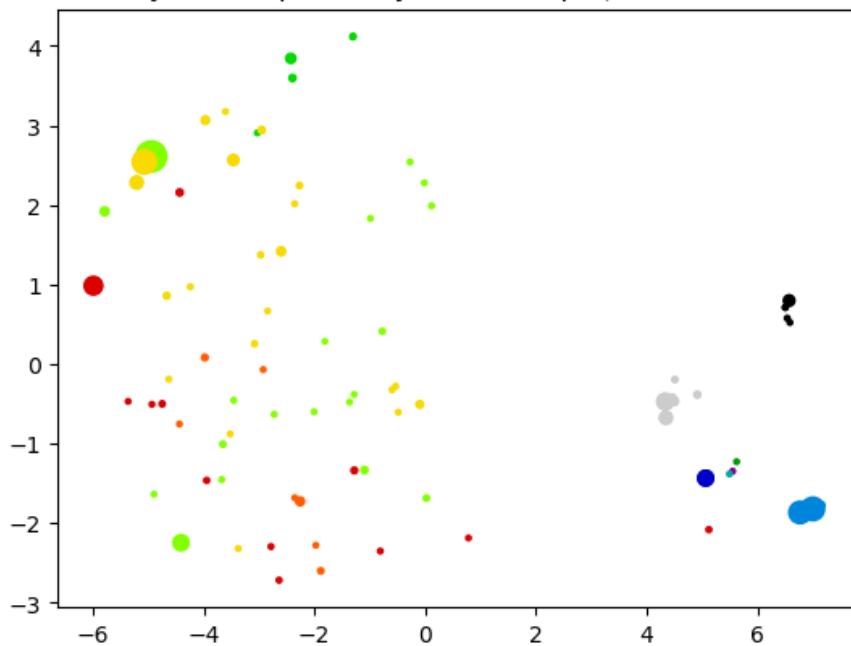
-----
TSNE(early_exaggeration=12, random_state=47, verbose=1)
-----
====> Finding 90 nearest neighbors using exact search using euclidean distance...
    --> Time elapsed: 0.00 seconds
====> Calculating affinity matrix...
    --> Time elapsed: 0.00 seconds
====> Calculating PCA-based initialization...
    --> Time elapsed: 0.00 seconds
====> Running optimization with exaggeration=12.00, lr=7.75 for 250 iterations...
Iteration 50, KL divergence 0.6582, 50 iterations in 0.0412 sec
Iteration 100, KL divergence 0.8738, 50 iterations in 0.0433 sec
Iteration 150, KL divergence 0.8757, 50 iterations in 0.0430 sec
Iteration 200, KL divergence 0.8634, 50 iterations in 0.0426 sec
Iteration 250, KL divergence 0.9332, 50 iterations in 0.0323 sec
    --> Time elapsed: 0.20 seconds
====> Running optimization with exaggeration=1.00, lr=93.00 for 500 iterations...
Iteration 50, KL divergence 0.2440, 50 iterations in 0.0353 sec
Iteration 100, KL divergence 0.2412, 50 iterations in 0.0345 sec
Iteration 150, KL divergence 0.2330, 50 iterations in 0.0344 sec
Iteration 200, KL divergence 0.2327, 50 iterations in 0.0350 sec
Iteration 250, KL divergence 0.2239, 50 iterations in 0.0357 sec
Iteration 300, KL divergence 0.2240, 50 iterations in 0.0362 sec
Iteration 350, KL divergence 0.2240, 50 iterations in 0.0353 sec
Iteration 400, KL divergence 0.2195, 50 iterations in 0.0352 sec
Iteration 450, KL divergence 0.2225, 50 iterations in 0.0343 sec
Iteration 500, KL divergence 0.2302, 50 iterations in 0.0345 sec
    --> Time elapsed: 0.35 seconds

```

(93, 2)

| | codeUnit | artifact | communityId | centrality | x | y |
|---|----------------------------------|-----------------|-------------|------------|----------|----------|
| 0 | org.axonframework.test | axon-test-4.9.3 | 0 | 0.081762 | 6.568582 | 0.798732 |
| 1 | org.axonframework.test.aggregate | axon-test-4.9.3 | 0 | 0.016509 | 6.583201 | 0.522860 |
| 2 | org.axonframework.test.matchers | axon-test-4.9.3 | 0 | 0.033417 | 6.574769 | 0.799313 |
| 3 | org.axonframework.test.saga | axon-test-4.9.3 | 0 | 0.016509 | 6.499787 | 0.712144 |
| 4 | org.axonframework.test.utils | axon-test-4.9.3 | 0 | 0.017834 | 6.534545 | 0.576816 |

Java Package positioned by their dependency relationships (HashGNN node embeddings + t-SNE)



2.5 Node Embeddings for Java Packages using node2vec

The results have been provided by the query filename: `../cypher/Node_EMBEDDINGS/Node_EMBEDDINGS_0a_Query_Calculated.cypher`

| | codeUnitName | shortCodeUnitName | projectName | communityId | centrality | embedding |
|---|----------------------------------|-------------------|-----------------|-------------|------------|---|
| 0 | org.axonframework.test | test | axon-test-4.9.3 | 0 | 0.081762 | [1.3577989339828491, 0.9424709677696228, -0.21... |
| 1 | org.axonframework.test.aggregate | aggregate | axon-test-4.9.3 | 0 | 0.016509 | [1.1961934566497803, 0.9530820250511169, -0.14... |
| 2 | org.axonframework.test.matchers | matchers | axon-test-4.9.3 | 0 | 0.033417 | [1.2953076362609863, 1.0387639999389648, -0.25... |
| 3 | org.axonframework.test.saga | saga | axon-test-4.9.3 | 0 | 0.016509 | [1.322131633758545, 0.9173240065574646, -0.108... |
| 4 | org.axonframework.test.utils | utils | axon-test-4.9.3 | 0 | 0.017834 | [1.261975884437561, 0.9061228036880493, -0.168... |

```

-----
TSNE(early_exaggeration=12, random_state=47, verbose=1)
-----
====> Finding 90 nearest neighbors using exact search using euclidean distance...
    --> Time elapsed: 0.01 seconds
====> Calculating affinity matrix...
    --> Time elapsed: 0.00 seconds
====> Calculating PCA-based initialization...
    --> Time elapsed: 0.00 seconds
====> Running optimization with exaggeration=12.00, lr=7.75 for 250 iterations...
Iteration 50, KL divergence -0.8743, 50 iterations in 0.0394 sec
Iteration 100, KL divergence 0.9815, 50 iterations in 0.0132 sec
Iteration 150, KL divergence 0.9815, 50 iterations in 0.0124 sec
Iteration 200, KL divergence 0.9815, 50 iterations in 0.0124 sec
Iteration 250, KL divergence 0.9815, 50 iterations in 0.0124 sec
    --> Time elapsed: 0.09 seconds
====> Running optimization with exaggeration=1.00, lr=93.00 for 500 iterations...
Iteration 50, KL divergence 0.1752, 50 iterations in 0.0395 sec
Iteration 100, KL divergence 0.1486, 50 iterations in 0.0533 sec
Iteration 150, KL divergence 0.1429, 50 iterations in 0.0368 sec
Iteration 200, KL divergence 0.1416, 50 iterations in 0.0338 sec
Iteration 250, KL divergence 0.1417, 50 iterations in 0.0329 sec
Iteration 300, KL divergence 0.1386, 50 iterations in 0.0332 sec
Iteration 350, KL divergence 0.1389, 50 iterations in 0.0323 sec
Iteration 400, KL divergence 0.1387, 50 iterations in 0.0325 sec
Iteration 450, KL divergence 0.1388, 50 iterations in 0.0325 sec
Iteration 500, KL divergence 0.1388, 50 iterations in 0.0336 sec
    --> Time elapsed: 0.36 seconds

```

(93, 2)

| | codeUnit | artifact | communityId | centrality | x | y |
|---|----------------------------------|-----------------|-------------|------------|-----------|----------|
| 0 | org.axonframework.test | axon-test-4.9.3 | 0 | 0.081762 | -6.777599 | 3.138911 |
| 1 | org.axonframework.test.aggregate | axon-test-4.9.3 | 0 | 0.016509 | -6.958078 | 3.183419 |
| 2 | org.axonframework.test.matchers | axon-test-4.9.3 | 0 | 0.033417 | -6.930376 | 3.235786 |
| 3 | org.axonframework.test.saga | axon-test-4.9.3 | 0 | 0.016509 | -6.727924 | 3.327193 |
| 4 | org.axonframework.test.utils | axon-test-4.9.3 | 0 | 0.017834 | -6.547554 | 3.267291 |

Java Package positioned by their dependency relationships (node2vec node embeddings + t-SNE)

