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. Typescript Modules

1.1 Generate Node Embeddings for Typescript Modules using Fast Random Projection (Fast RP)

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	/home/runner/work/code-graph-analysis-pipeline...	react-router	react-router	0	0.507805	[0.01841725967824459, 0.08843754976987839, -0....
1	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	react-router-dom	0	0.221261	[0.020978230983018875, 0.0889035165309906, -0....
2	/home/runner/work/code-graph-analysis-pipeline...	react-router-native	react-router-native	0	0.190845	[0.031069429591298103, 0.06349994242191315, -0...
3	/home/runner/work/code-graph-analysis-pipeline...	server	react-router-dom	1	0.190845	[0.015934694558382034, 0.11374668776988983, -0...
4	/home/runner/work/code-graph-analysis-pipeline...	router	router	1	0.787613	[0.019688423722982407, 0.0865434780716896, -0....

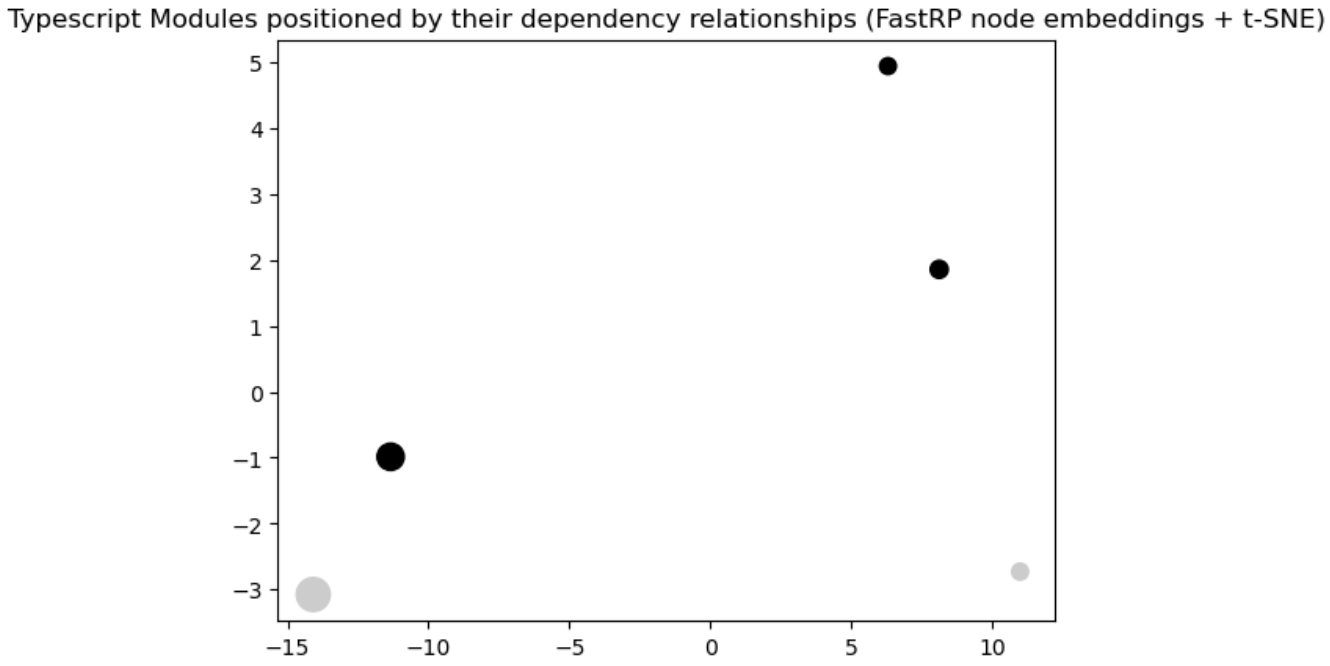
1.2 Dimensionality reduction with t-distributed stochastic neighbor embedding (t-SNE)

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".

```
Perplexity value 30 is too high. Using perplexity 1.33 instead
--------------------------------------------------------------------------------
TSNE(early_exaggeration=12, random_state=47, verbose=1)
--------------------------------------------------------------------------------
===> Finding 4 nearest neighbors using exact search using euclidean distance...
   --> Time elapsed: 0.03 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=0.42 for 250 iterations...
Iteration   50, KL divergence 0.9821, 50 iterations in 0.0069 sec
Iteration  100, KL divergence 1.2053, 50 iterations in 0.0065 sec
Iteration  150, KL divergence 1.2053, 50 iterations in 0.0061 sec
Iteration  200, KL divergence 1.2053, 50 iterations in 0.0061 sec
Iteration  250, KL divergence 1.2053, 50 iterations in 0.0061 sec
   --> Time elapsed: 0.03 seconds
===> Running optimization with exaggeration=1.00, lr=5.00 for 500 iterations...
Iteration   50, KL divergence 0.0728, 50 iterations in 0.0065 sec
Iteration  100, KL divergence 0.0724, 50 iterations in 0.0067 sec
Iteration  150, KL divergence 0.0718, 50 iterations in 0.0065 sec
Iteration  200, KL divergence 0.0713, 50 iterations in 0.0065 sec
Iteration  250, KL divergence 0.0708, 50 iterations in 0.0064 sec
Iteration  300, KL divergence 0.0704, 50 iterations in 0.0064 sec
Iteration  350, KL divergence 0.0701, 50 iterations in 0.0065 sec
Iteration  400, KL divergence 0.0697, 50 iterations in 0.0207 sec
Iteration  450, KL divergence 0.0694, 50 iterations in 0.0094 sec
Iteration  500, KL divergence 0.0691, 50 iterations in 0.0069 sec
   --> Time elapsed: 0.08 seconds
(5, 2)
```

	codeUnit	artifact	communityId	centrality	x	y
0	/home/runner/work/code-graph-analysis-pipeline...	react-router	0	0.507805	-11.344739	-0.987607
1	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	0	0.221261	8.127205	1.855159
2	/home/runner/work/code-graph-analysis-pipeline...	react-router-native	0	0.190845	6.309217	4.938178
3	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	1	0.190845	10.998107	-2.729356
4	/home/runner/work/code-graph-analysis-pipeline...	router	1	0.787613	-14.089790	-3.076374

1.3 Plot the node embeddings reduced to two dimensions for Typescript



1.4 Node Embeddings for Typescript Modules 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.

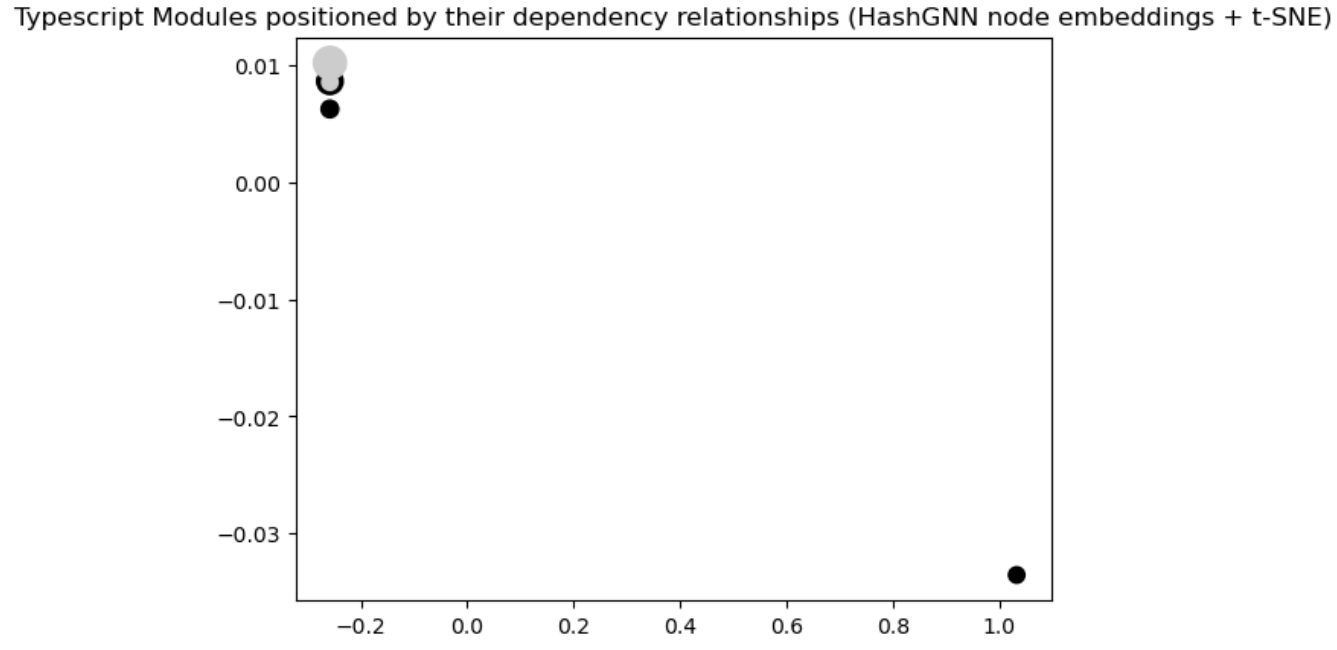
```
The results have been provided by the query filename: ../cypher/Node_Embeddings/Node_
Embeddings_0a_Query_Calculated.cypher
```

	codeUnitName	shortCodeUnitName	projectName	communityId	centrality	embedding
0	/home/runner/work/code-graph-analysis-pipeline...	react-router	react-router	0	0.507805	[0.9185586869716644, 0.9185586869716644, 0.0, ...
1	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	react-router-dom	0	0.221261	[0.9185586869716644, 0.9185586869716644, 0.0, ...
2	/home/runner/work/code-graph-analysis-pipeline...	react-router-native	react-router-native	0	0.190845	[0.9185586869716644, 0.9185586869716644, 0.0, ...
3	/home/runner/work/code-graph-analysis-pipeline...	server	react-router-dom	1	0.190845	[0.9185586869716644, 0.9185586869716644, 0.0, ...
4	/home/runner/work/code-graph-analysis-pipeline...	router	router	1	0.787613	[0.9185586869716644, 0.9185586869716644, 0.0, ...

```
Perplexity value 30 is too high. Using perplexity 1.33 instead
```

```
--------------------------------------------------------------------------------
TSNE(early_exaggeration=12, random_state=47, verbose=1)
--------------------------------------------------------------------------------
===> Finding 4 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=0.42 for 250 iterations...
Iteration   50, KL divergence -0.7070, 50 iterations in 0.0067 sec
Iteration  100, KL divergence 0.3147, 50 iterations in 0.0063 sec
Iteration  150, KL divergence 0.3147, 50 iterations in 0.0062 sec
Iteration  200, KL divergence 0.3147, 50 iterations in 0.0062 sec
Iteration  250, KL divergence 0.3147, 50 iterations in 0.0062 sec
   --> Time elapsed: 0.03 seconds
===> Running optimization with exaggeration=1.00, lr=5.00 for 500 iterations...
Iteration   50, KL divergence 0.0064, 50 iterations in 0.0068 sec
Iteration  100, KL divergence 0.0000, 50 iterations in 0.0071 sec
Iteration  150, KL divergence 0.0000, 50 iterations in 0.0073 sec
Iteration  200, KL divergence 0.0000, 50 iterations in 0.0071 sec
Iteration  250, KL divergence 0.0000, 50 iterations in 0.0070 sec
Iteration  300, KL divergence 0.0000, 50 iterations in 0.0071 sec
Iteration  350, KL divergence 0.0000, 50 iterations in 0.0070 sec
Iteration  400, KL divergence 0.0000, 50 iterations in 0.0072 sec
Iteration  450, KL divergence 0.0000, 50 iterations in 0.0071 sec
Iteration  500, KL divergence 0.0000, 50 iterations in 0.0071 sec
   --> Time elapsed: 0.07 seconds
(5, 2)
```

	codeUnit	artifact	communityId	centrality	x	y
0	/home/runner/work/code-graph-analysis-pipeline...	react-router	0	0.507805	-0.258057	0.008611
1	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	0	0.221261	-0.258133	0.006238
2	/home/runner/work/code-graph-analysis-pipeline...	react-router-native	0	0.190845	1.032255	-0.033553
3	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	1	0.190845	-0.258060	0.008524
4	/home/runner/work/code-graph-analysis-pipeline...	router	1	0.787613	-0.258005	0.010180



1.5 Node Embeddings for Typescript Modules using node2vec

node2vec computes a vector representation of a node based on second order random walks in the graph. The node2vec algorithm is a transductive node embedding algorithm, meaning that it needs the whole graph to be available to learn the node embeddings.

```
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	/home/runner/work/code-graph-analysis-pipeline...	react-router	react-router	0	0.507805	[0.040191248059272766, -0.2577258348464966, 0....
1	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	react-router-dom	0	0.221261	[0.03887362778186798, -0.24269945919513702, 0....
2	/home/runner/work/code-graph-analysis-pipeline...	react-router-native	react-router-native	0	0.190845	[0.03648390248417854, -0.23585890233516693, 0....
3	/home/runner/work/code-graph-analysis-pipeline...	server	react-router-dom	1	0.190845	[0.036411091685295105, -0.24583780765533447, 0...
4	/home/runner/work/code-graph-analysis-pipeline...	router	router	1	0.787613	[0.03292982280254364, -0.22829119861125946, 0....

```
Perplexity value 30 is too high. Using perplexity 1.33 instead
--------------------------------------------------------------------------------
TSNE(early_exaggeration=12, random_state=47, verbose=1)
--------------------------------------------------------------------------------
===> Finding 4 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=0.42 for 250 iterations...
Iteration   50, KL divergence 0.6434, 50 iterations in 0.0069 sec
Iteration  100, KL divergence 0.9719, 50 iterations in 0.0068 sec
Iteration  150, KL divergence 0.9719, 50 iterations in 0.0073 sec
Iteration  200, KL divergence 0.9719, 50 iterations in 0.0068 sec
Iteration  250, KL divergence 0.9719, 50 iterations in 0.0070 sec
   --> Time elapsed: 0.03 seconds
===> Running optimization with exaggeration=1.00, lr=5.00 for 500 iterations...
Iteration   50, KL divergence 0.1716, 50 iterations in 0.0068 sec
Iteration  100, KL divergence 0.1699, 50 iterations in 0.0070 sec
Iteration  150, KL divergence 0.1696, 50 iterations in 0.0070 sec
Iteration  200, KL divergence 0.1692, 50 iterations in 0.0067 sec
Iteration  250, KL divergence 0.1688, 50 iterations in 0.0067 sec
Iteration  300, KL divergence 0.1685, 50 iterations in 0.0068 sec
Iteration  350, KL divergence 0.1683, 50 iterations in 0.0065 sec
Iteration  400, KL divergence 0.1681, 50 iterations in 0.0153 sec
Iteration  450, KL divergence 0.1680, 50 iterations in 0.0065 sec
Iteration  500, KL divergence 0.1679, 50 iterations in 0.0064 sec
   --> Time elapsed: 0.08 seconds
(5, 2)
```

	codeUnit	artifact	communityId	centrality	x	y
0	/home/runner/work/code-graph-analysis-pipeline...	react-router	0	0.507805	11.112752	10.492029
1	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	0	0.221261	-9.019636	0.527357
2	/home/runner/work/code-graph-analysis-pipeline...	react-router-native	0	0.190845	4.665914	0.520817
3	/home/runner/work/code-graph-analysis-pipeline...	react-router-dom	1	0.190845	-16.989965	0.129530
4	/home/runner/work/code-graph-analysis-pipeline...	router	1	0.787613	10.230935	-11.669734

Typescript Modules positioned by their dependency relationships (node2vec node embeddings + t-SNE)