Mining Networks

Chapter 6 — Graph Embeddings

Bogumił Kamiński, Paweł Prałat, and François Théberge

Updated: 2026/01/05

Department of Mathematics, Toronto Metropolitan University

File: DS8014-Chapter06

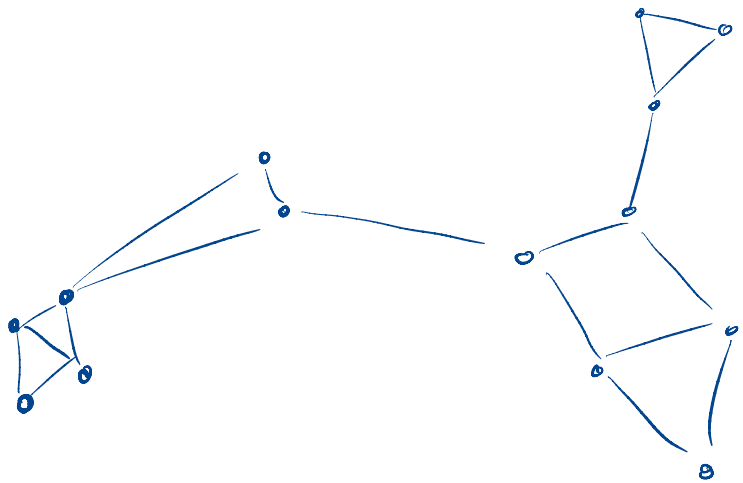
Toronto Metropolitan University

Overview

1. Problem Formalization
2. Techniques
3. Unsupervised Benchmark
4. A Few Applications
5. Experiments

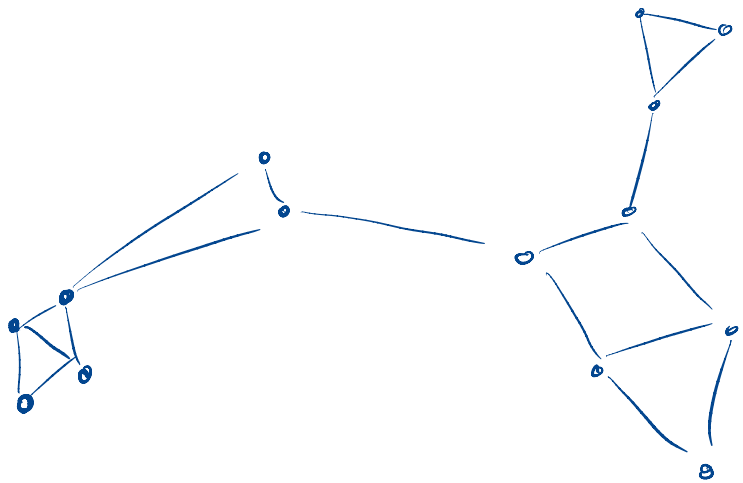
Introduction

To extract useful structural information from graphs, it is convenient to embed it in a geometric space by assigning coordinates to each node.



Introduction

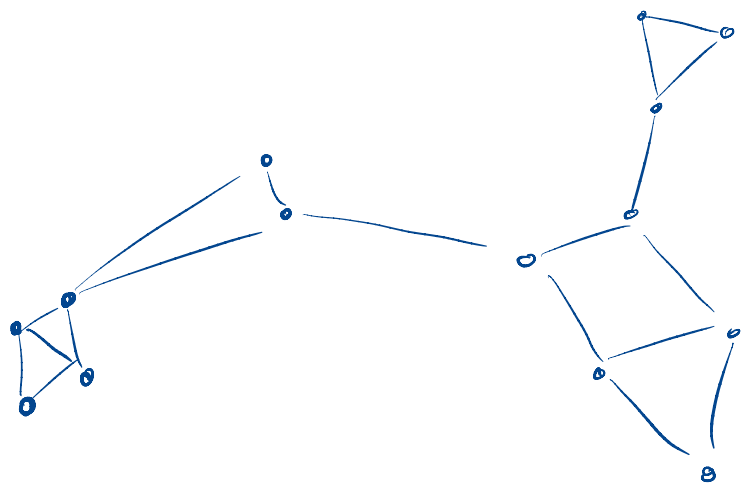
Classical embeddings: nearby nodes are more likely to share an edge than those far from each other, or are similar in some sense.



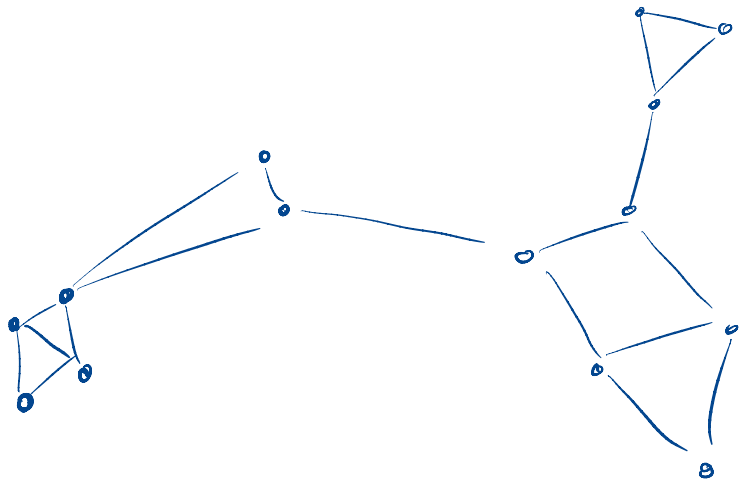
Introduction

Classical embeddings: nearby nodes are more likely to share an edge than those far from each other, or are similar in some sense.

Structural embeddings: nodes with similar structure of their local ego-nets should be embedded close to each other.



Introduction



There are many applications, including:

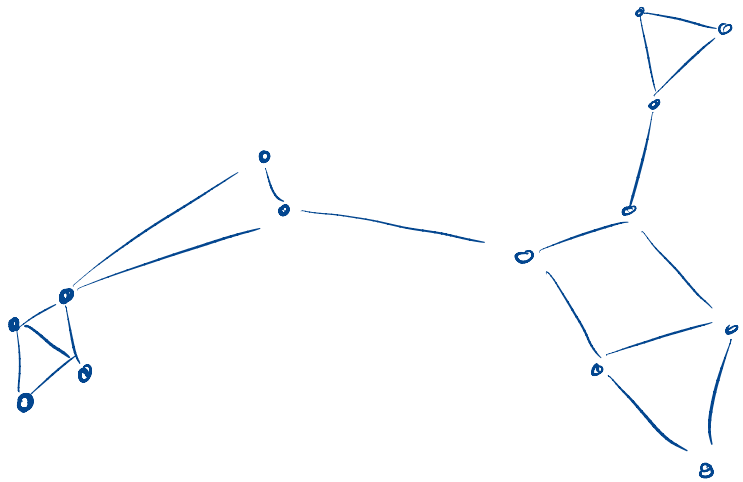
– Node Classification (say, bot detection): structural,

– Node Clustering and Community Detection: classical,

– Link Prediction and Missing Links: classical,

– Visualization.

Introduction



There are many applications, including:

– Node Classification (say, bot detection): structural,

– Node Clustering and Community Detection: classical,

– Link Prediction and Missing Links: classical,

– Visualization.

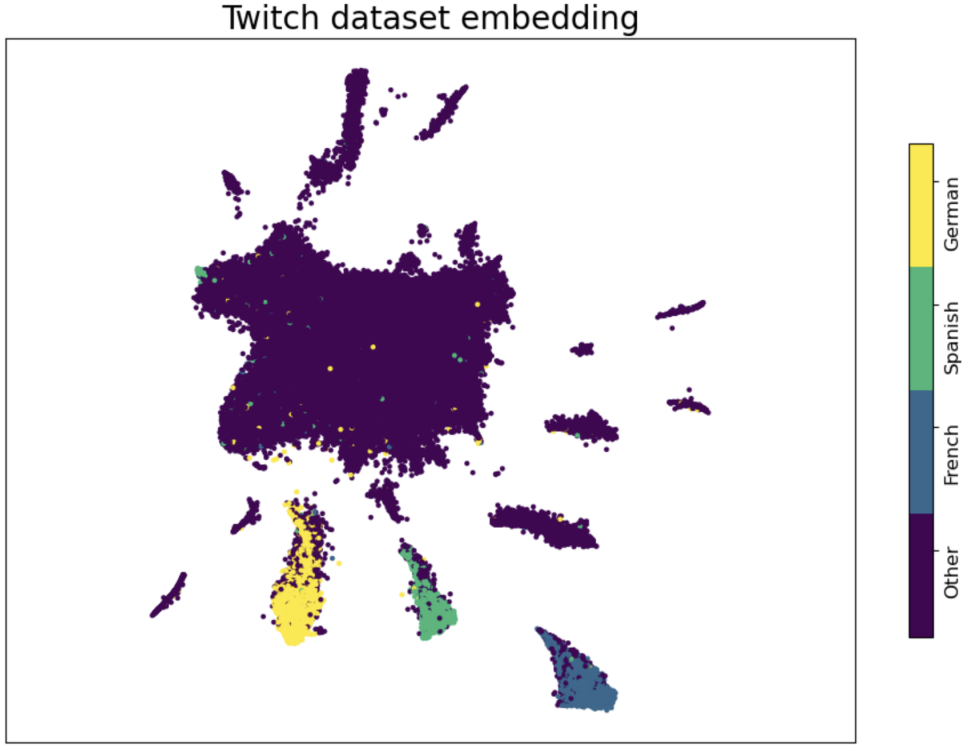
Drawback: graph embedding often requires domain experts.

Illustration

Twitch gamers social network:

– 168,114 nodes representing users

– 6,797,557 (undirected) edges — mutual followers

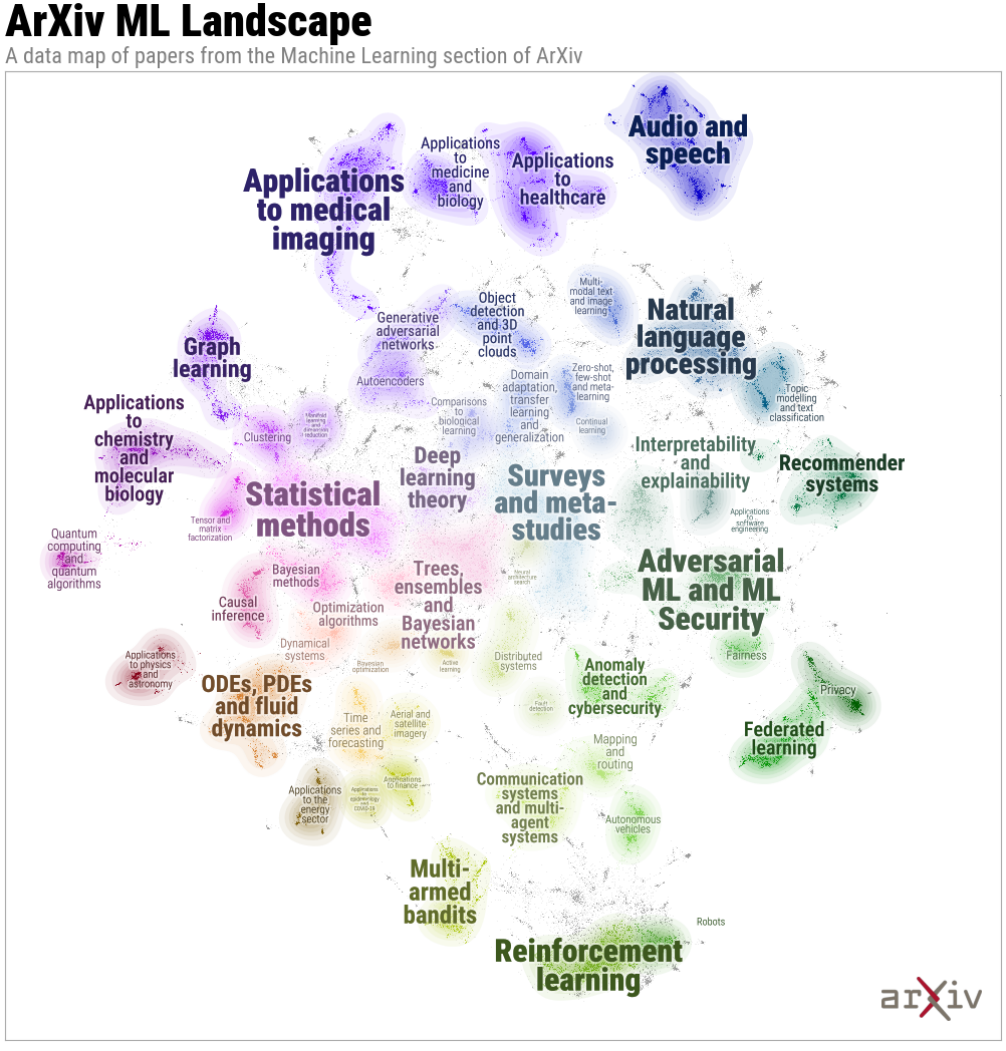


Node2Vec + UMAP

Illustration

ArXiv ML Landscape

(Example of DataMapPlot in action)



Problem Formalization

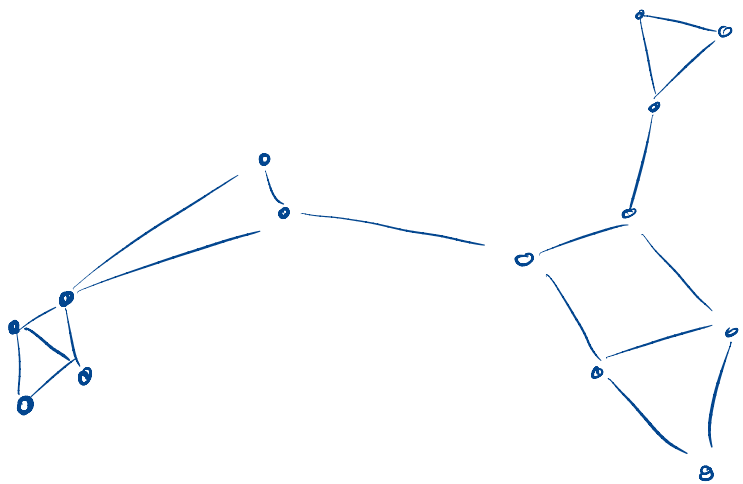
Embedding

$G = (V, E)$ — weighted undirected graph on the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$. (We will discuss directed graphs later on.)

Embedding

$G = (V, E)$ — weighted undirected graph on the set of nodes $V = \{v_1, v_2, \ldots, v_n\}$. (We will discuss directed graphs later on.)

Embedding — function $\mathcal{E} \colon V \to \mathbb{R}^k$; k is much smaller than n.



Embedding

$\mathscr{E}$ decreases the dimension but at the same time it tries to preserve pairwise proximity between nodes as best as possible.

Embedding

$\mathscr{E}$ decreases the dimension but at the same time it tries to preserve pairwise proximity between nodes as best as possible.

There are a few natural proximity measures, each produces a matrix $\mathbf{S} = (s(v_i, v_j))_{i,j \in [n]}$.
$s(v_i, v_j) \in \mathbb{R}_+ \cup \{0\}$ — proximity between nodes v_i and v_j.

Embedding

$\mathscr{E}$ decreases the dimension but at the same time it tries to preserve pairwise proximity between nodes as best as possible.

There are a few natural proximity measures, each produces a matrix $\mathbf{S} = (s(v_i, v_j))_{i,j \in [n]}$.
$s(v_i, v_j) \in \mathbb{R}_+ \cup \{0\}$ — proximity between nodes v_i and v_j.

It is desired that matrix $\mathbf{S}$ is symmetric but not all proximity measures guarantee this property. (Embedding algorithm has to deal with asymmetric matrices if needed.)
The diagonal of $\mathbf{S}$ is usually ignored by embedding algorithms but it does not have to be.

First-order Proximity

Two nodes are simply more similar if they are connected by an edge with larger weight.

First-order Proximity

The first-order proximity $s_1(v_i, v_j)$ is the weight of the edge $v_i v_j$, that is, $s_1(v_i, v_j) = a(v_i, v_j)$, where $\mathbf{A} = (a(v_i, v_j))_{i,j \in [n]}$ is the adjacency matrix.

Second-order Proximity

It measures the similarity between the neighbourhoods.

Second-order Proximity

The second-order proximity $s_2(v_i, v_j)$ is a similarity between v_i's neighbourhood

$$s_1(v_i) = (s_1(v_i, v_1), s_1(v_i, v_2), \ldots, s_1(v_i, v_n))$$

and v_j's neighbourhood

$$s_1(v_j) = (s_1(v_j, v_1), s_1(v_j, v_2), \ldots, s_1(v_j, v_n)).$$

Second-order Proximity

It measures the similarity between the neighbourhoods.

Second-order Proximity

The second-order proximity $s_2(v_i, v_j)$ is a similarity between v_i's neighbourhood and v_j's neighbourhood.

The cosine similarity, a standard measure of similarity, is defined as the cosine of the angle between them:

$$s_2(v_i, v_j) = \frac{\sum_{\ell=1}^{n} s_1(v_i, v_\ell)\, s_1(v_j, v_\ell)}{\sqrt{\sum_{\ell=1}^{n} s_1(v_i, v_\ell)^2}\, \sqrt{\sum_{\ell=1}^{n} s_1(v_j, v_\ell)^2}}.$$

Second-order Proximity

It measures the similarity between the neighbourhoods.

Second-order Proximity

The second-order proximity $s_2(v_i, v_j)$ is a similarity between v_i's neighbourhood and v_j's neighbourhood.

The cosine similarity, a standard measure of similarity, is defined as the cosine of the angle between them:

$$s_2(v_i, v_j) = \frac{\sum_{\ell=1}^{n} s_1(v_i, v_\ell)\, s_1(v_j, v_\ell)}{\sqrt{\sum_{\ell=1}^{n} s_1(v_i, v_\ell)^2}\, \sqrt{\sum_{\ell=1}^{n} s_1(v_j, v_\ell)^2}}.$$

Generalization: for any $k \in \mathbb{N} \setminus \{1, 2\}$, the kth-order proximity $s_k(v_i, v_j)$ is the similarity between v_i's $(k-1)$st neighbourhood $s_{k-1}(v_i)$ and v_j's $(k-1)$st neighbourhood $s_{k-1}(v_j)$.

Katz Index

Alternatively, one may want to use some centrality measure such as the Katz centrality. It tries to capture the relative influence of a node within a network by considering walks of any length but penalize longer walks by introducing the attenuation factor α.

Katz Index

Fix any α such that $0 < \alpha < \min\{1, 1/|\lambda|\}$, where λ is the leading eigenvalue of adjacency matrix $\mathbf{A}$. The Katz Index $\mathbf{S}_\alpha^{\text{Katz}}$ is defined as follows:

$$\mathbf{S}_\alpha^{\text{Katz}} = \sum_{i=1}^{\infty} (\alpha \cdot A)^i = (\mathbf{I}/\alpha - \mathbf{A})^{-1}\mathbf{A}.$$

Personalized PageRank

This variant (commonly used by web search engines) of the well-known PageRank depends on two parameters: the jumping constant α and the seed node $s \in V$.

Random walk continues with probability α and goes back to the seed node s with probability $1 - \alpha$.
(In PageRank a seed is uniformly sampled from V.)

Personalized PageRank

This variant (commonly used by web search engines) of the well-known PageRank depends on two parameters: the jumping constant α and the seed node $s \in V$.

Random walk continues with probability α and goes back to the seed node s with probability $1 - \alpha$.
(In PageRank a seed is uniformly sampled from V.)

Personalized PageRank

Fix any $0 < \alpha < 1$. The Personalized PageRank $\mathbf{S}_\alpha^{\mathrm{PPR}}$ is defined as follows:

$$\mathbf{S}_\alpha^{\mathrm{PPR}} = (1-\alpha)\left(\mathbf{I} - \alpha\hat{\mathbf{A}}^T\right)^{-1},$$

where the auxiliary matrix $\hat{\mathbf{A}}$ is defined as in PageRank.

Personalized PageRank

This variant (commonly used by web search engines) of the well-known PageRank depends on two parameters: the jumping constant α and the seed node $s \in V$.

Random walk continues with probability α and goes back to the seed node s with probability $1 - \alpha$.
(In PageRank a seed is uniformly sampled from V.)

Personalized PageRank

Fix any $0 < \alpha < 1$. The Personalized PageRank $\mathbf{S}_\alpha^{\mathrm{PPR}}$ is defined as follows:

$$\mathbf{S}_\alpha^{\mathrm{PPR}} = (1-\alpha)\left(\mathbf{I} - \alpha\hat{\mathbf{A}}^T\right)^{-1},$$

where the auxiliary matrix $\hat{\mathbf{A}}$ is defined as in PageRank.

$\mathbf{S}_\alpha^{\mathrm{PPR}}$ is usually not symmetric.

Common Neighbours

In the next definition, we simply count the number of nodes that have both v_i and v_j as their neighbours.

Common Neighbours

Given adjacency matrix $\mathbf{A}$, the Common Neighbours $\mathbf{S}^{\text{CN}}$ is defined as follows:

$$\mathbf{S}^{\text{CN}} = \mathbf{A}^2.$$

Recall that: $\mathbf{A}^k$ is equal to the number of walks of length k from node v_i to node v_j.

Adamic-Adar

Variant of the common neighbours in which the fact that two nodes are common neighbours of some node with very large neighbourhood is less significant.

Adamic-Adar

The Adamic-Adar $\mathbf{S}^{\mathrm{AA}}$ is defined as follows:

$$s^{AA}(v_i, v_j) = \sum_{v_k \in N(v_i) \cap N(v_j)} \frac{1}{\ln(\deg(v_k))},$$

for $i \neq j$ and $s^{AA}(v_i, v_i) = 0$.

Directed Graphs

- Compute **S** using the original (asymmetric) matrix **A**.

Directed Graphs

- Compute $\mathbf{S}$ using the original (asymmetric) matrix $\mathbf{A}$.

- Consider $(\mathbf{A} + \mathbf{A}^T)/2$ instead (replace two directed edges between v_i and v_j by a single undirected one with the weight equal to the average weight of the two). Some proximity measures might produce an asymmetric matrix $\mathbf{S}$ anyway!

Directed Graphs

- Compute $\mathbf{S}$ using the original (asymmetric) matrix $\mathbf{A}$.

- Consider $(\mathbf{A} + \mathbf{A}^T)/2$ instead (replace two directed edges between v_i and v_j by a single undirected one with the weight equal to the average weight of the two). Some proximity measures might produce an asymmetric matrix $\mathbf{S}$ anyway!

If this happens, then we have two options:
– ignore it and let the embedding algorithm to deal with this (for example, **LLE** and **HOPE** can do it),
– pass $(\mathbf{S} + \mathbf{S}^T)/2$ instead of $\mathbf{S}$ to an embedding algorithm.

Directed Graphs

- Compute **S** using the original (asymmetric) matrix **A**.

- Consider $(\mathbf{A} + \mathbf{A}^T)/2$ instead (replace two directed edges between v_i and v_j by a single undirected one with the weight equal to the average weight of the two). Some proximity measures might produce an asymmetric matrix **S** anyway!

If this happens, then we have two options:
– ignore it and let the embedding algorithm to deal with this (for example, **LLE** and **HOPE** can do it),
– pass $(\mathbf{S} + \mathbf{S}^T)/2$ instead of **S** to an embedding algorithm.

There are also some proximity measures (for example, SimRank that is similar to second-order measures) that can be explicitly used to compute **S** for directed graphs.

Techniques

Linear Algebra Algorithms

Local Linear Embedding (LLE):

This algorithm works for both undirected and directed graphs.

Local Linear Embedding (LLE):

This algorithm works for both undirected and directed graphs.

We assume that for all nodes v_i, $\deg^{out}(v_i) \neq 0$.

Local Linear Embedding (LLE):

This algorithm works for both undirected and directed graphs.

We assume that for all nodes v_i, $\deg^{out}(v_i) \neq 0$.

Let $e_i = \mathcal{E}(v_i)$ be the embedding of node v_i, and let $\mathbf{E}$ be the $k \times n$ matrix consisting of vectors e_i (they form the columns).

Local Linear Embedding (LLE):

This algorithm works for both undirected and directed graphs.

We assume that for all nodes v_i, $\deg^{out}(v_i) \neq 0$.

Let $e_i = \mathcal{E}(v_i)$ be the embedding of node v_i, and let $\mathbf{E}$ be the $k \times n$ matrix consisting of vectors e_i (they form the columns).

Let

$$\hat{a}(u, v) = a(u, v)/\deg^{out}(u).$$

Local Linear Embedding (LLE):

This algorithm works for both undirected and directed graphs.

We assume that for all nodes v_i, $\deg^{out}(v_i) \neq 0$.

Let $e_i = \mathcal{E}(v_i)$ be the embedding of node v_i, and let $\mathbf{E}$ be the $k \times n$ matrix consisting of vectors e_i (they form the columns).

Let

$$\hat{a}(u, v) = a(u, v)/\deg^{out}(u).$$

e_i should be close to

$$\sum_{v_j \in N^{out}(v_i)} \hat{a}(v_i, v_j) e_j = \sum_{j=1}^{n} \hat{a}(v_i, v_j) e_j.$$

Local Linear Embedding (LLE):

The goal is then to minimize the following optimization function:

$$\Phi(\mathbf{E}) = \left\|(I - \hat{\mathbf{A}})\mathbf{E}^T\right\|_F,$$

where $\|\cdot\|_F$ is the Frobenius norm, a natural extension of the Euclidean norm to matrices:

$$\|\mathbf{B}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} b_{i,j}^2}.$$

Local Linear Embedding (LLE):

Since $\mathbf{E} = 0_{k \times n}$ is a valid solution to such unconstrained problem, we need to add some additional conditions that will allow us to find a non-degenerate embedding matrix $\mathbf{E}$:

Local Linear Embedding (LLE):

Since $\mathbf{E} = 0_{k \times n}$ is a valid solution to such unconstrained problem, we need to add some additional conditions that will allow us to find a non-degenerate embedding matrix $\mathbf{E}$:

– $\mathbf{E1} = \mathbf{0}$, that is, rows of $\mathbf{E}$ are centred around the origin,

Local Linear Embedding (LLE):

Since $\mathbf{E} = 0_{k \times n}$ is a valid solution to such unconstrained problem, we need to add some additional conditions that will allow us to find a non-degenerate embedding matrix $\mathbf{E}$:

– $\mathbf{E1} = \mathbf{0}$, that is, rows of $\mathbf{E}$ are centred around the origin,

– $\mathbf{E}\mathbf{E}^T/n = I$, that is, rows of $\mathbf{E}$ have equal norms and are mutually orthogonal; knowing one of the dimensions of the embedding provides no information about the missing dimensions.

Local Linear Embedding (LLE):

Since $\mathbf{E} = 0_{k \times n}$ is a valid solution to such unconstrained problem, we need to add some additional conditions that will allow us to find a non-degenerate embedding matrix $\mathbf{E}$:

– $\mathbf{E1} = \mathbf{0}$, that is, rows of $\mathbf{E}$ are centred around the origin,

– $\mathbf{E}\mathbf{E}^T/n = I$, that is, rows of $\mathbf{E}$ have equal norms and are mutually orthogonal; knowing one of the dimensions of the embedding provides no information about the missing dimensions.

Result: there is a unique solution (up to the rotation).

Linear Algebra Algorithms

Local Linear Embedding (LLE):

...

$$\Phi(\mathbf{E}) = \mathrm{tr}(\mathbf{E}(I - \hat{\mathbf{A}})^T(I - \hat{\mathbf{A}})\mathbf{E}^T)$$

Linear Algebra Algorithms

Local Linear Embedding (LLE):

...

$$\Phi(\mathbf{E}) = \text{tr}(\mathbf{E}(I - \hat{\mathbf{A}})^T(I - \hat{\mathbf{A}})\mathbf{E}^T)$$

...

$\Phi(\mathbf{E})$ is minimized for $\mathbf{E}$ consisting of rows taken as k eigenvectors corresponding to smallest eigenvalues of $(I - \hat{\mathbf{A}})^T(I - \hat{\mathbf{A}})$, excluding the smallest eigenvalue that is equal to 0 (which produces a non-centred eigenvector as discussed above).

Laplacian Eigenmaps (LEM):

Uses the graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

This time our goal is to minimize the following optimization function:

$$\Phi(\mathbf{E}) = \sum_{(v_i, v_j) \in V^2} \left\| e_i - e_j \right\|^2 a(v_i, v_j) = \mathrm{tr}(\mathbf{E}\mathbf{L}\mathbf{E}^T),$$

that is, points that are connected by heavily weighted edges should be close to each other in the embedded space.

Laplacian Eigenmaps (LEM):

As before, the problem can be reduced to finding k eigenvectors corresponding to smallest eigenvalues of $\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$, excluding the smallest eigenvalue that is equal to 0 (the 0 eigenvalue always exists and has multiplicity of one, provided the graph is connected).

Laplacian Eigenmaps (LEM):

As before, the problem can be reduced to finding k eigenvectors corresponding to smallest eigenvalues of $\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$, excluding the smallest eigenvalue that is equal to 0 (the 0 eigenvalue always exists and has multiplicity of one, provided the graph is connected).

Finally, note that in this procedure one can replace $\mathbf{A}$ by any proximity measure $\mathbf{S}$ as long as it is symmetric.

Linear Algebra Algorithms

Another general approach is to find a suitable embedding that minimizes the following loss function:

$$\Phi(\mathbf{E}) = \left\| \mathbf{S} - \mathbf{E}^T \mathbf{E} \right\|_F ,$$

where $\mathbf{S}$ is some proximity measure matrix. (Keeping the inner product of the embedding vectors of two given nodes as close as possible to the similarity between the nodes in the graph.)

Linear Algebra Algorithms

Another general approach is to find a suitable embedding that minimizes the following loss function:

$$\Phi(\mathbf{E}) = \left\| \mathbf{S} - \mathbf{E}^T \mathbf{E} \right\|_F ,$$

where $\mathbf{S}$ is some proximity measure matrix. (Keeping the inner product of the embedding vectors of two given nodes as close as possible to the similarity between the nodes in the graph.)

High Order Proximity preserved Embedding (HOPE):

Specific algorithm designed for directed graphs.

$$\Phi(\mathbf{E}_s, \mathbf{E}_t) = \left\| \mathbf{S} - \mathbf{E}_s^T \mathbf{E}_t \right\|_F ,$$

where $\mathbf{E}_s$ and $\mathbf{E}_t$ are the corresponding matrices of the source and the target embeddings.

Algorithms Based on Random Walk

These methods are inspired by the Word2Vec algorithm for word embedding in Natural Language Processing (NLP).

Main idea: words are known by the company they keep.

Algorithms Based on Random Walk

These methods are inspired by the Word2Vec algorithm for word embedding in Natural Language Processing (NLP).

Main idea: words are known by the company they keep.

Uses the model known as SkipGram. Consider a context window of size 5 (that is, we take $\ell = 2$) and the sentence:

"Graph *embedding maps* **nodes** *to vector* space".

Algorithms Based on Random Walk

These methods are inspired by the Word2Vec algorithm for word embedding in Natural Language Processing (NLP).

Main idea: words are known by the company they keep.

Uses the model known as SkipGram. Consider a context window of size 5 (that is, we take $\ell = 2$) and the sentence:

"Graph *embedding maps* **nodes** *to vector* space".

The model is trained to predict the words in *italics* given the word "**nodes**" as input.

Algorithms Based on Random Walk

Similarly to HOPE, SkipGram associates two embeddings with each word: $\mathbf{E}_s$ ("source") and $\mathbf{E}_t$ ("target").

They try to capture relationships between the input words and the words that they aim to predict and vice versa.

Algorithms Based on Random Walk

Similarly to HOPE, SkipGram associates two embeddings with each word: $\mathbf{E}_s$ ("source") and $\mathbf{E}_t$ ("target").

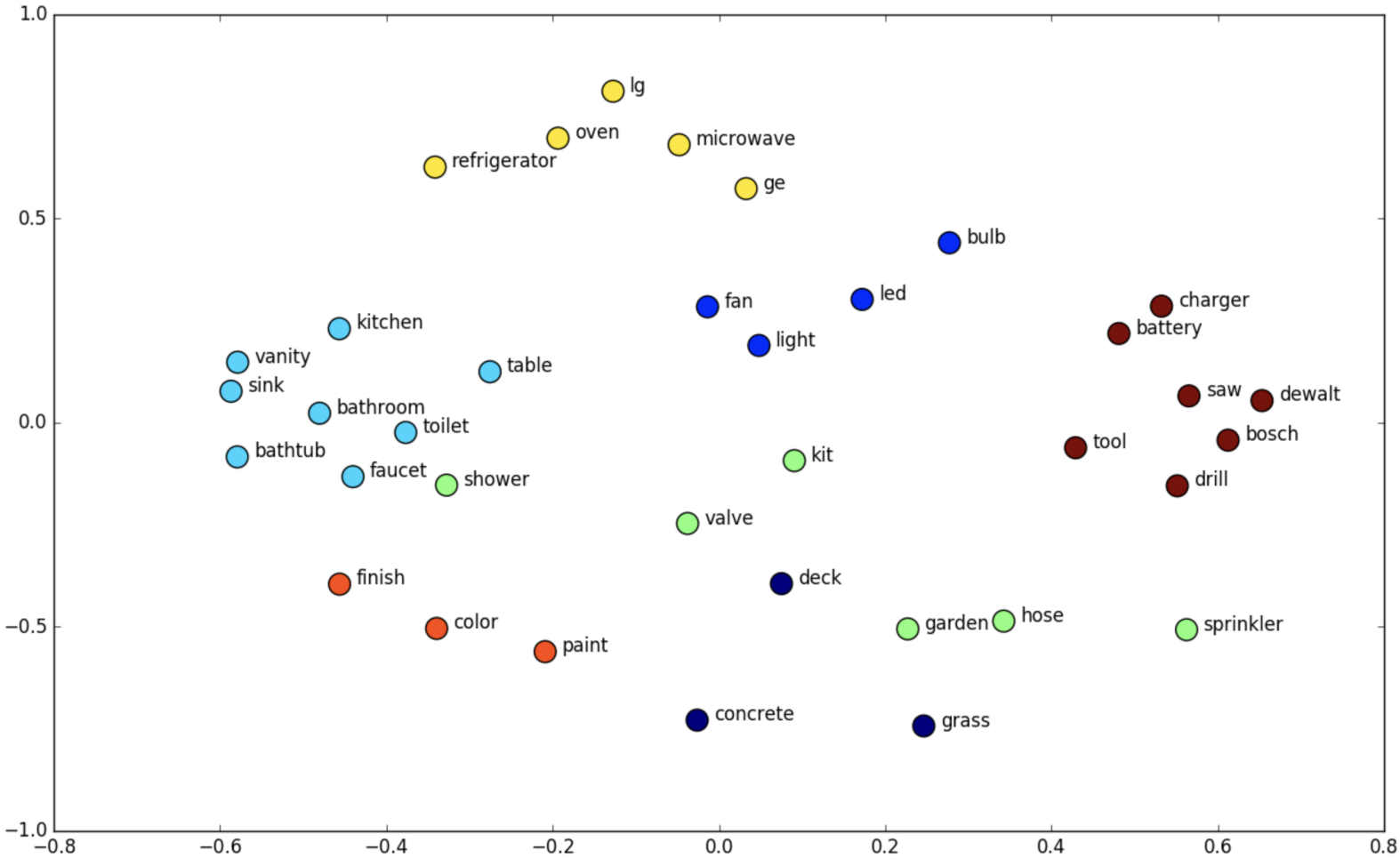
They try to capture relationships between the input words and the words that they aim to predict and vice versa.

Given a word i, the probability $p_{i,j}$ that we see word j in its neighbourhood is approximated by the softmax function:

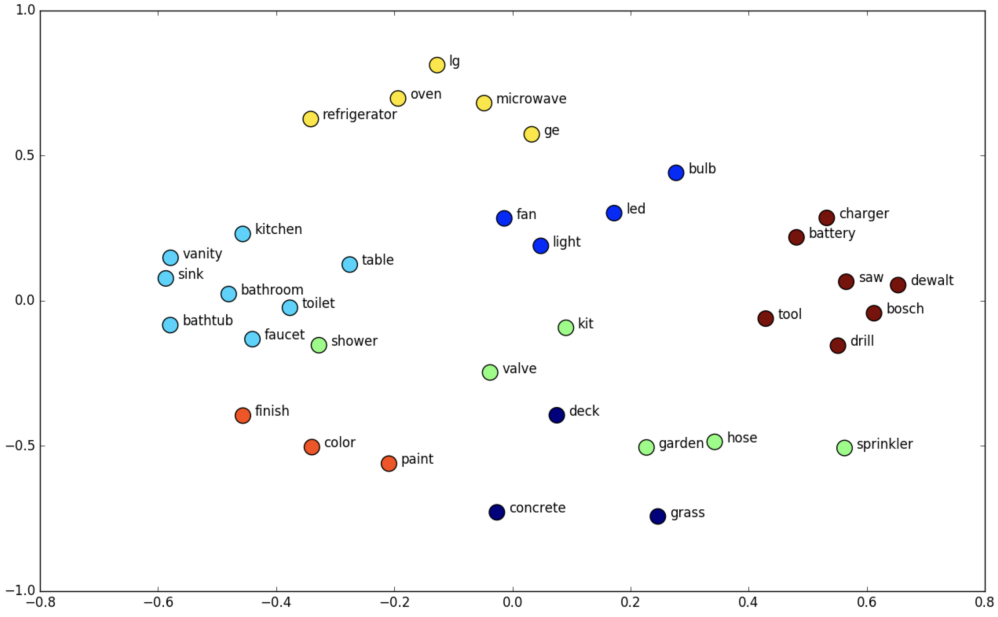
$$q_{i,j} = \frac{\exp(e_{t,j}^T e_{s,i})}{\sum_{\ell=1}^{|W|} \exp(e_{t,\ell}^T e_{s,i})}.$$

The model is trained using maximum likelihood estimation.

Algorithms Based on Random Walk



Algorithms Based on Random Walk



Unsupervised learning (no syntactic or semantic relationships among words are provided).

Yet, word embeddings seem to capture these relationships!

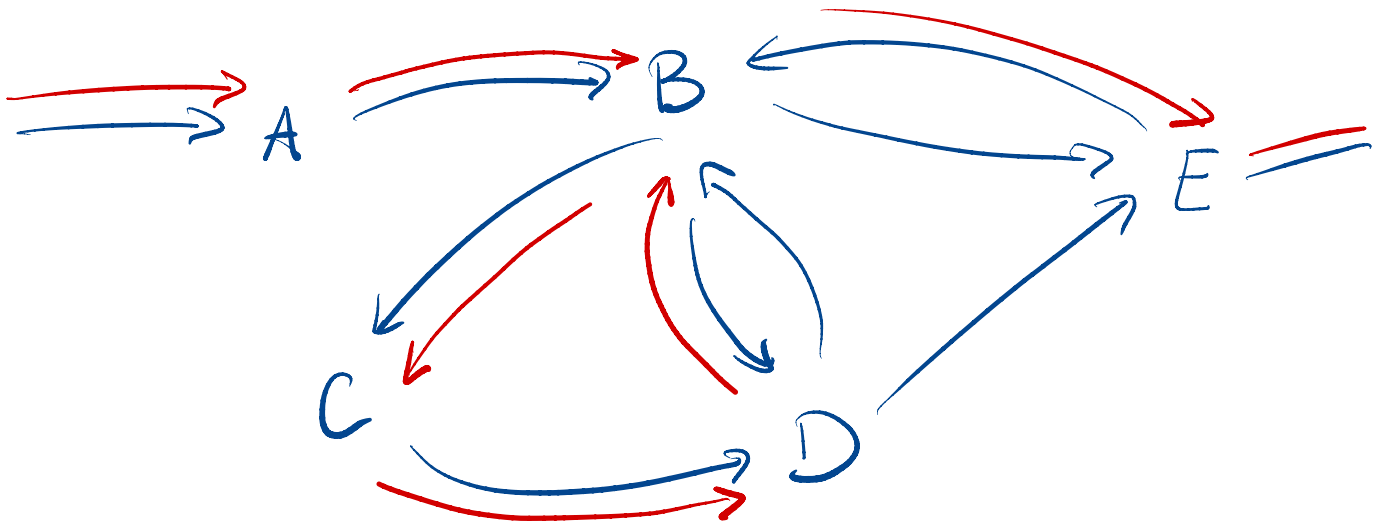
king - man + woman ≈ queen

Paris - France + Germany ≈ Berlin

Algorithms Based on Random Walk

Generalization to graphs.

– the words are simply the nodes of a graph,
– sentences (sequences of nodes) are generated via random walks on a graph (exact procedure depends on the algorithm).



Algorithms Based on Random Walk

Main challenge: the number of nodes is usually much larger than the number of words in the typical NLP application. (Computing the denominator in $q_{i,j}$ has complexity $\Theta(n)$.)

Algorithms Based on Random Walk

Main challenge: the number of nodes is usually much larger than the number of words in the typical NLP application. (Computing the denominator in $q_{i,j}$ has complexity $\Theta(n)$.)

Solution: use a predictive model called hierarchical softmax. Nodes are represented as leaves in a binary tree (Huffman tree); a binary classifier is fitted at each split. As a result, we only need to evaluate $\Theta(\ln n)$ classifies.

Algorithms Based on Random Walk

Deep Walk algorithm:

– Generate random walks of some fixed length, typically between 32 and 64 per node.

Algorithms Based on Random Walk

Deep Walk algorithm:

– Generate random walks of some fixed length, typically between 32 and 64 per node.

– Depending of the context window size, we explore node's neighbourhoods beyond the first and the second order proximity.

Algorithms Based on Random Walk

Node2Vec algorithm:

Depending on the parameterization, the random walks are respectively biased toward the following two extremes:
– Breadth-First Search (BFS): walks tend to stay near the initial node and so they mimic the BFS algorithm
(they tend to preserve the macro-view),
– Depth-First Search (DFS): walks tend to move away from the initial node and so they behave as the DFS algorithm
(they tend to preserve the micro-view).

The user may smoothly move between these two extremes by changing the parameters of Node2Vec, as required in a given application.

Deep Learning Algorithms

Autoencoders:

Encoder $E : \mathbf{R}^n \to \mathbf{R}^k$ is an ℓ-layer neural network that takes n-dimensional input and produces k-dimensional embedding.

Deep Learning Algorithms

Autoencoders:

Encoder $E\colon \mathbf{R}^n \to \mathbf{R}^k$ is an ℓ-layer neural network that takes n-dimensional input and produces k-dimensional embedding.

Decoder is a function $D\colon \mathbf{R}^k \to \mathbf{R}^n$.

Deep Learning Algorithms

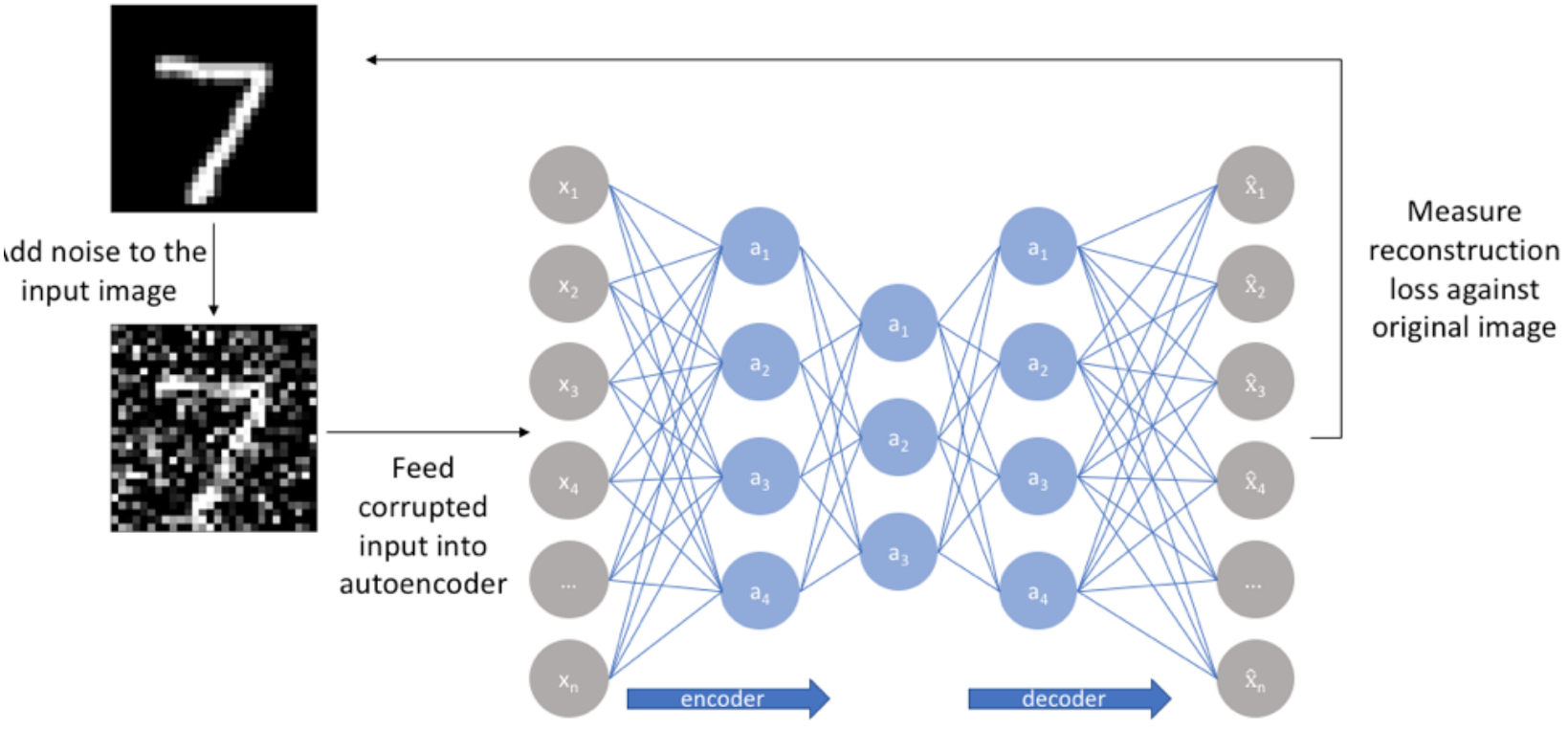
Autoencoders:

Encoder $E\colon \mathbf{R}^n \to \mathbf{R}^k$ is an ℓ-layer neural network that takes n-dimensional input and produces k-dimensional embedding.

Decoder is a function $D\colon \mathbf{R}^k \to \mathbf{R}^n$.

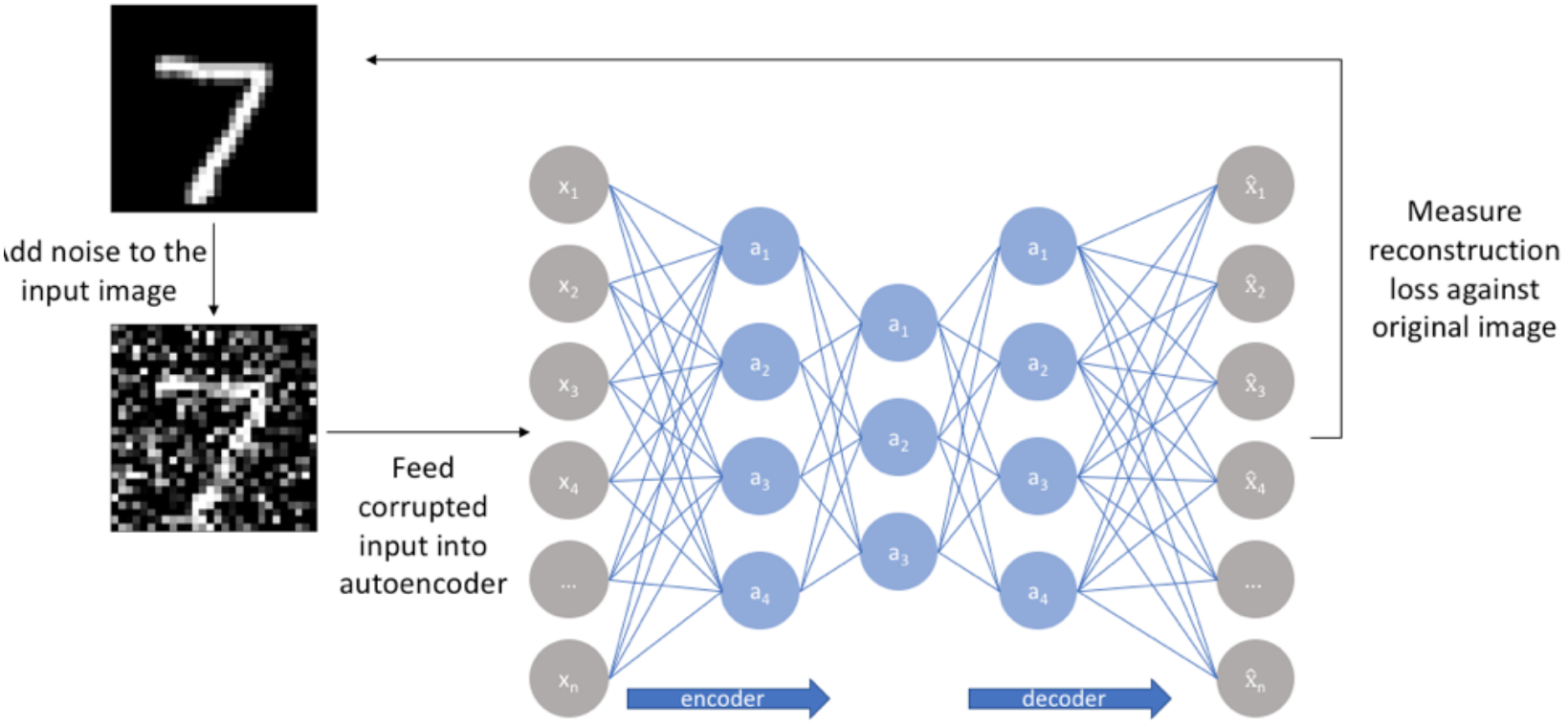
Both function form the autoencoder that tries to get $D(E(a_i))$ as close to a_i as possible.... well, not only that.

Deep Learning Algorithms



Autoencoders train the network to ignore "noise".

Deep Learning Algorithms



Autoencoders train the network to ignore "noise".

Structural Deep Network Embedding (SDNE) is an example of such family of algorithms.

Structural Node Embeddings

Graph Convolution Network (GCN): see the book

Recursive Feature Extraction (ReFeX):

– f_i^0 features associated with node v_i: external metadata or some property of the node such as its degree, clustering coefficient, etc. (f^0 is a vector of vectors.)

Structural Node Embeddings

Graph Convolution Network (GCN): see the book

Recursive Feature Extraction (ReFeX):

– f_i^0 features associated with node v_i: external metadata or some property of the node such as its degree, clustering coefficient, etc. (f^0 is a vector of vectors.)

In each step,
– aggregate using some function $a(\cdot)$ (e.g. mean, sum):
$f_i^s = a(\{f_j^{s-1} : j \in N(v_i)\})$;

Structural Node Embeddings

Graph Convolution Network (GCN): see the book

Recursive Feature Extraction (ReFeX):

– f_i^0 features associated with node v_i: external metadata or some property of the node such as its degree, clustering coefficient, etc. (f^0 is a vector of vectors.)

In each step,
– aggregate using some function $a(\cdot)$ (e.g. mean, sum):
$f_i^s = a(\{f_j^{s-1} : j \in N(v_i)\})$;
– prune: select a subset I^s of features from
$\{f^j : j \in \{0, 1, \ldots, s\}\}$ that are not redundant.

Structural Node Embeddings

Graph Convolution Network (GCN): see the book

Recursive Feature Extraction (ReFeX):

– f_i^0 features associated with node v_i: external metadata or some property of the node such as its degree, clustering coefficient, etc. (f^0 is a vector of vectors.)

In each step,
– aggregate using some function $a(\cdot)$ (e.g. mean, sum):
$f_i^s = a(\{f_j^{s-1} : j \in N(v_i)\})$;
– prune: select a subset I^s of features from
$\{f^j : j \in \{0, 1, \ldots, s\}\}$ that are not redundant.

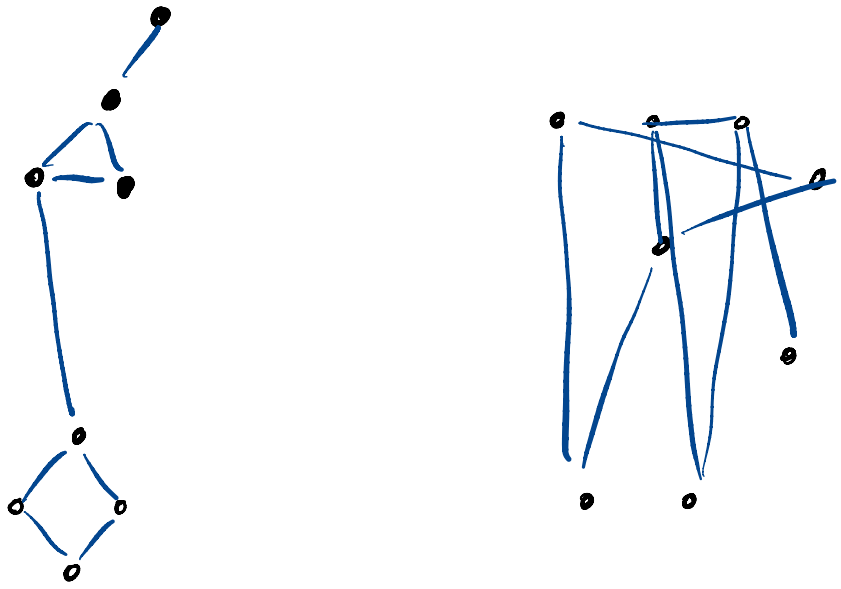
Repeat t times or until no new features are introduced. Vectors from $\bigcup_{j=0}^{t} I^j$ form the embedding.

Unsupervised Benchmark

Main Goal

There are many embedding algorithms (techniques from linear algebra, random walks, or deep learning) and the list grows.

Results vary a lot.



How can we evaluate these embeddings? Which one is the best and should be used? Important question: GIGO.

Big Picture

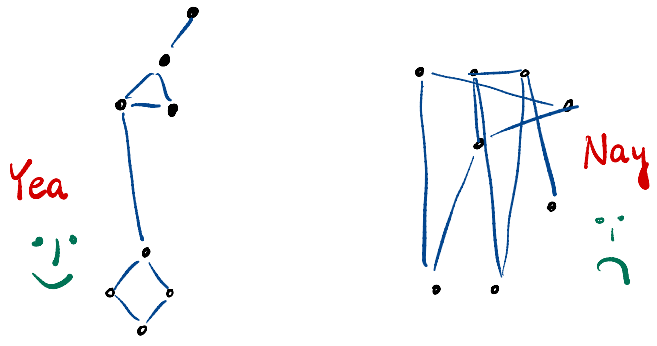
Input: Graph $G = (V, E)$ on n nodes with the degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$.

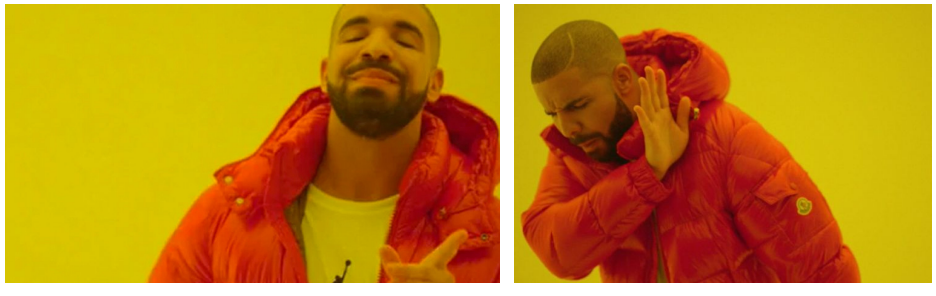
Embedding of nodes of V, $\mathscr{E} : V \to \mathbb{R}^k$ (typically many).

Big Picture

Input: Graph $G = (V, E)$ on n nodes with the degree distribution $\mathbf{w} = (w_1, \ldots, w_n)$.
Embedding of nodes of V, $\mathscr{E} : V \to \mathbb{R}^k$ (typically many).

Output: two "divergence scores" (global and local) assigned to $\mathscr{E}$ (smaller is better).





Global Score

Global score:

– looks at the network and the associated embeddings “from the distance”;

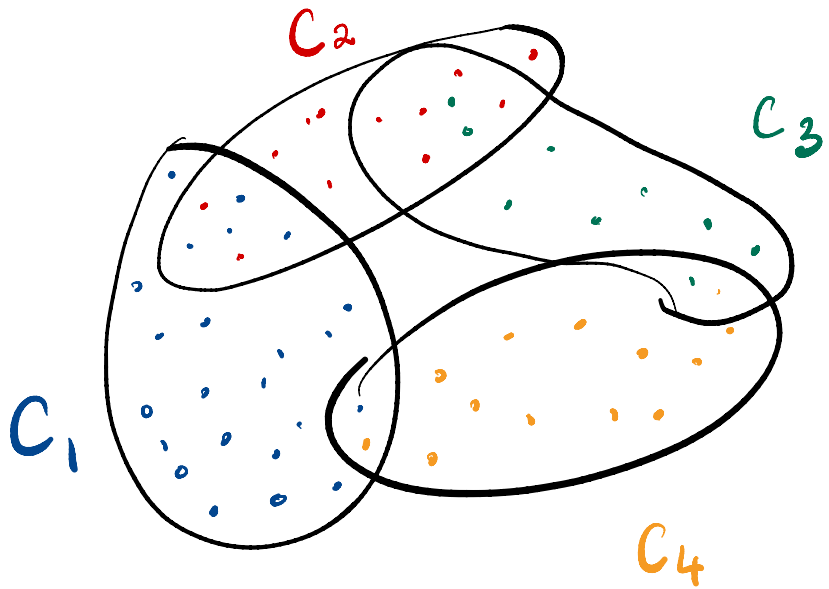
– evaluates the embeddings based on their ability to capture global properties of the network, namely, edge densities;

Global Score

Global score:
– looks at the network and the associated embeddings "from the distance";
– evaluates the embeddings based on their ability to capture global properties of the network, namely, edge densities;

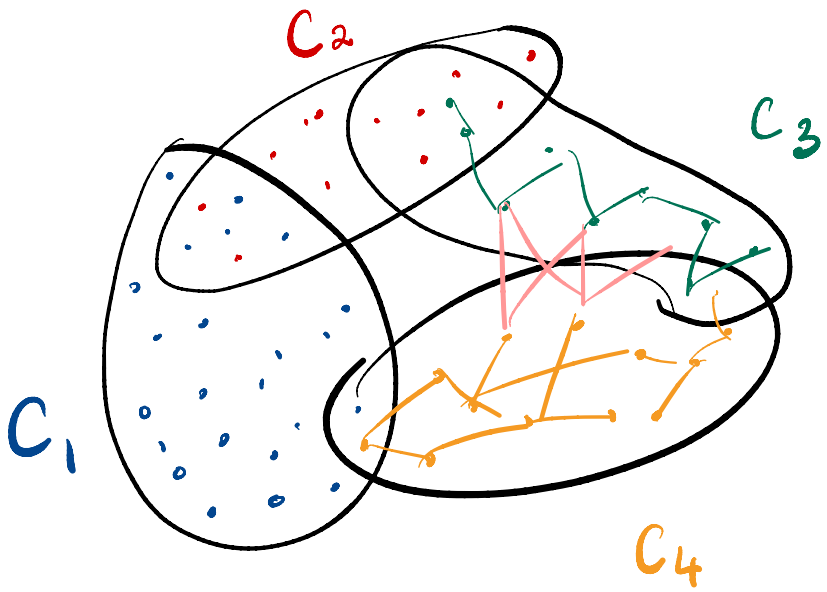
– goodness-of-fit test for the provided embedding, rather than simply checking its predictive power—its aim is similar to the well-known Hosmer–Lemeshow test for logistic regression (used frequently in risk prediction models).

Global Score



Step 1: Run some graph clustering algorithm to obtain a partition of V into ℓ communities $C_1, \ldots, C_\ell$.

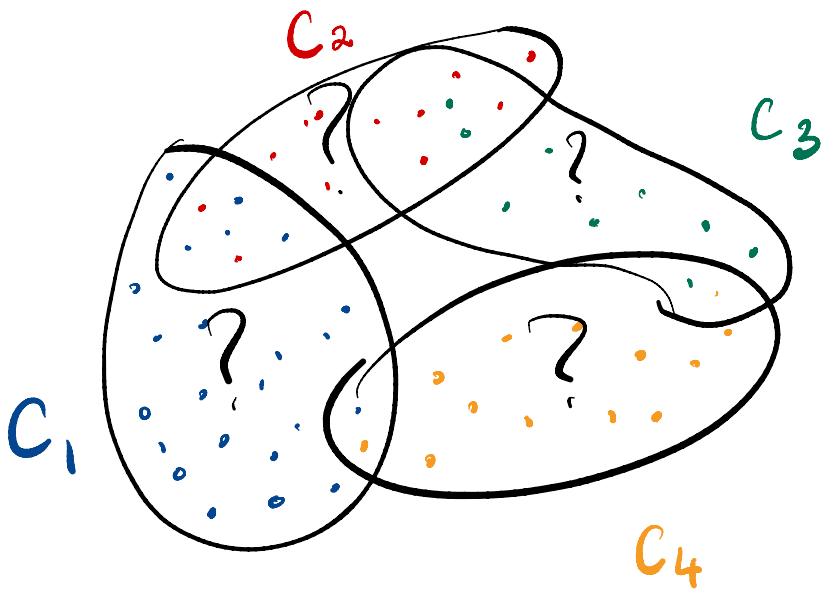
Global Score



Step 2: Compute $c_{i,j}$ (including $j = i$): proportion of edges with one endpoint in C_i and the other one in C_j.

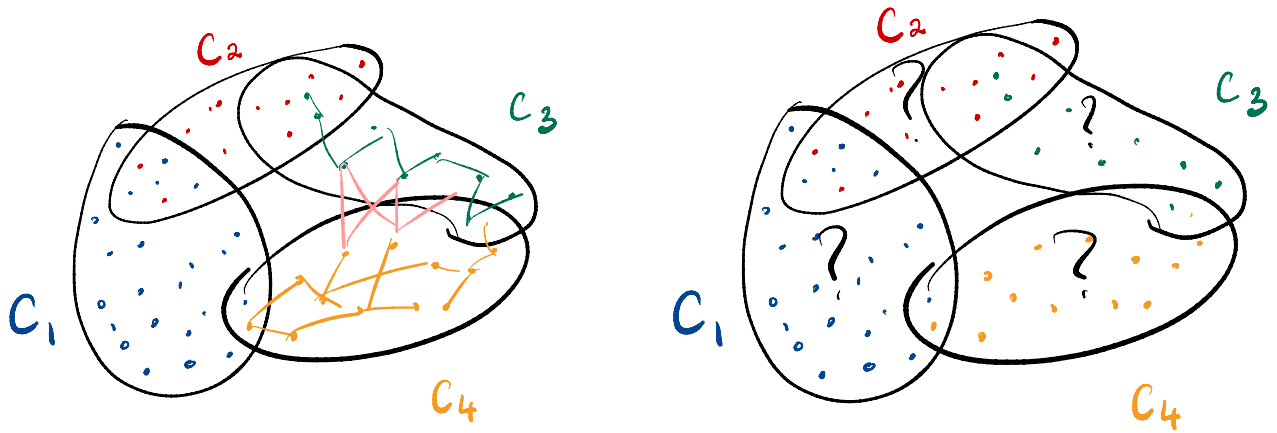
Note that we do not use $\mathscr{E}$.

Global Score



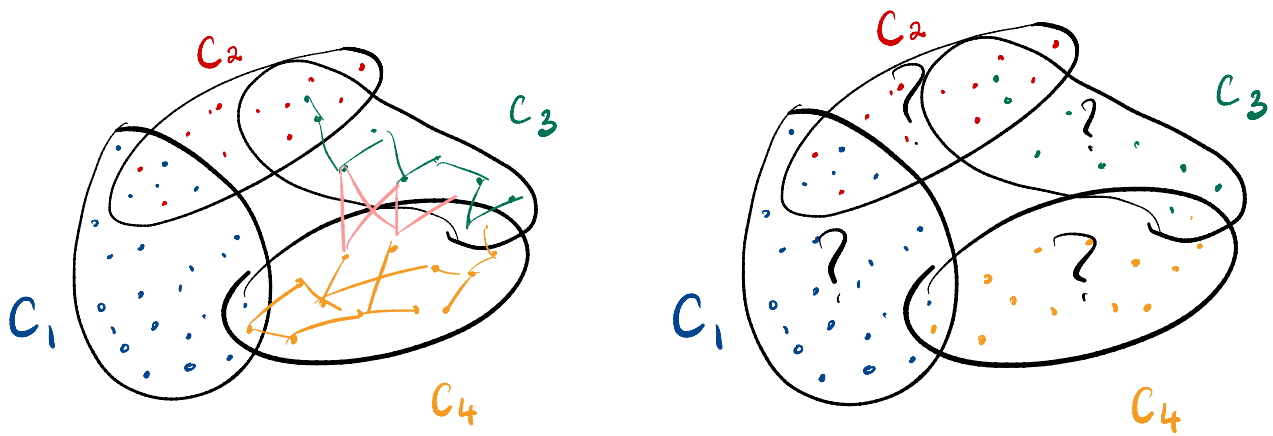
Step 3: Compute $b_{i,j}$ (including $j = i$): the expected proportion of edges with one endpoint in C_i and the other one in C_j, in the geometric Chung-Lu model $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$.
Note that we do not use G, only $\mathbf{w}$ (on top of $\mathcal{E}$ and parameter α, the "aversion" for long links).

Global Score



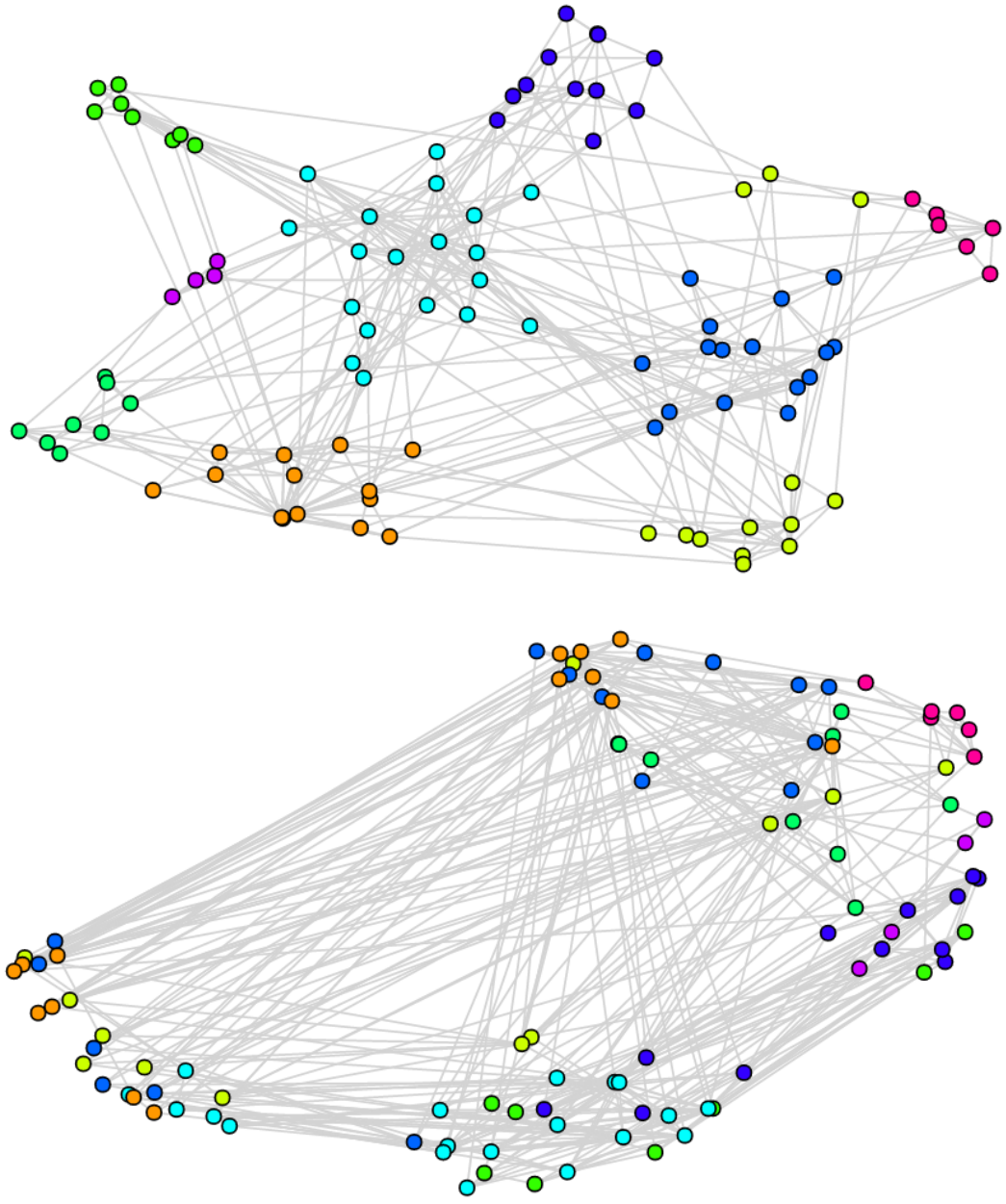
Step 4: Compute Δ_α: the Jensen-Shannon divergence between the two vectors (smoothed version of the Kullback-Leibler divergence we mentioned earlier).

Global Score

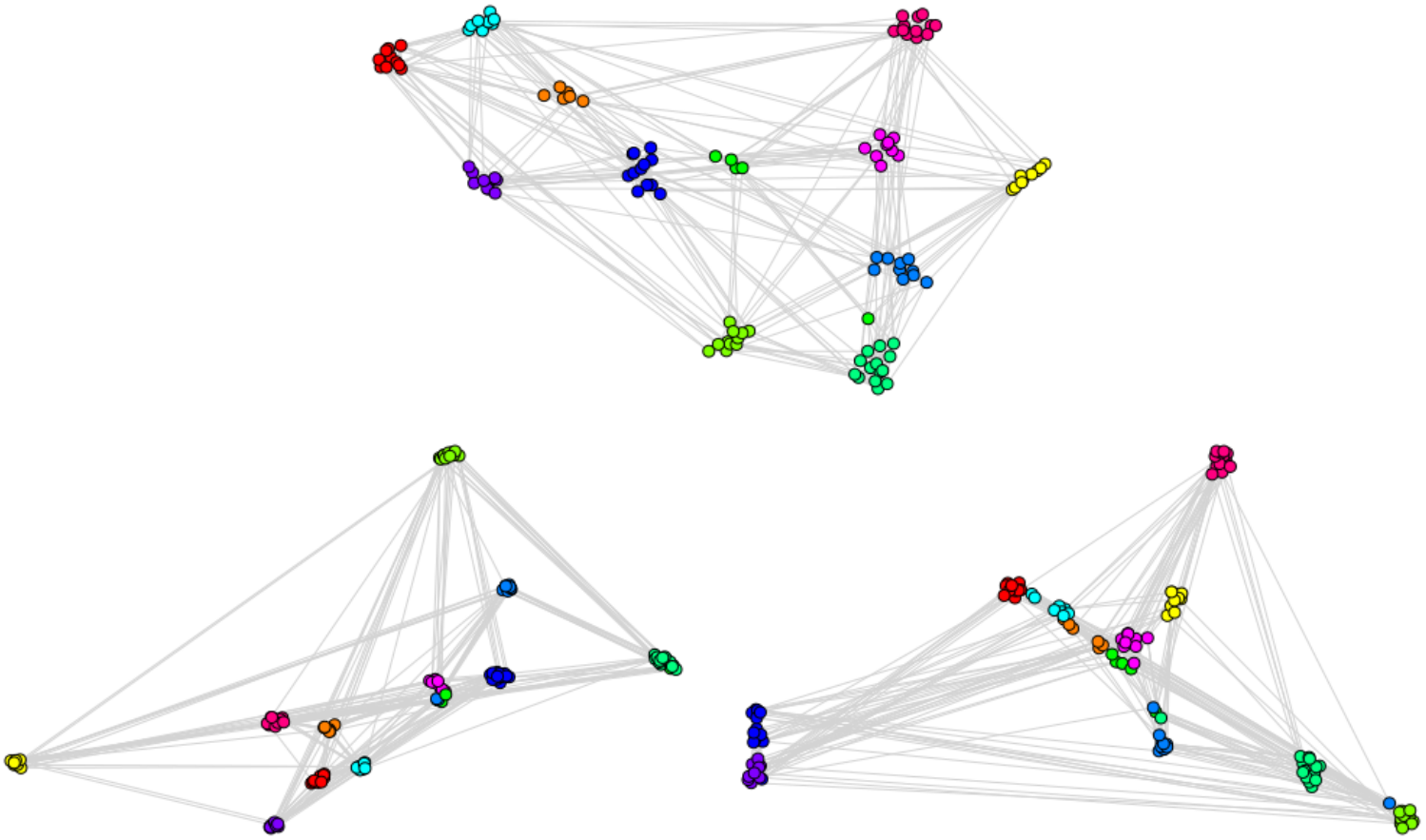


Step 5: Take the minimum Δ_α over various choices for parameter α that measures the "aversion" for long links. It defines the global score.

The worst and the best LFR graph ($\mu = 0.35$)



The worst and the best College Football graph



Local Score

Local score:
– provides a complementary test,
– looks at the network and embeddings "under the microscope", trying to see if a local structure of a graph is well reflected by the associated embedding.

Local Score

Local score:
– provides a complementary test,
– looks at the network and embeddings "under the microscope", trying to see if a local structure of a graph is well reflected by the associated embedding.

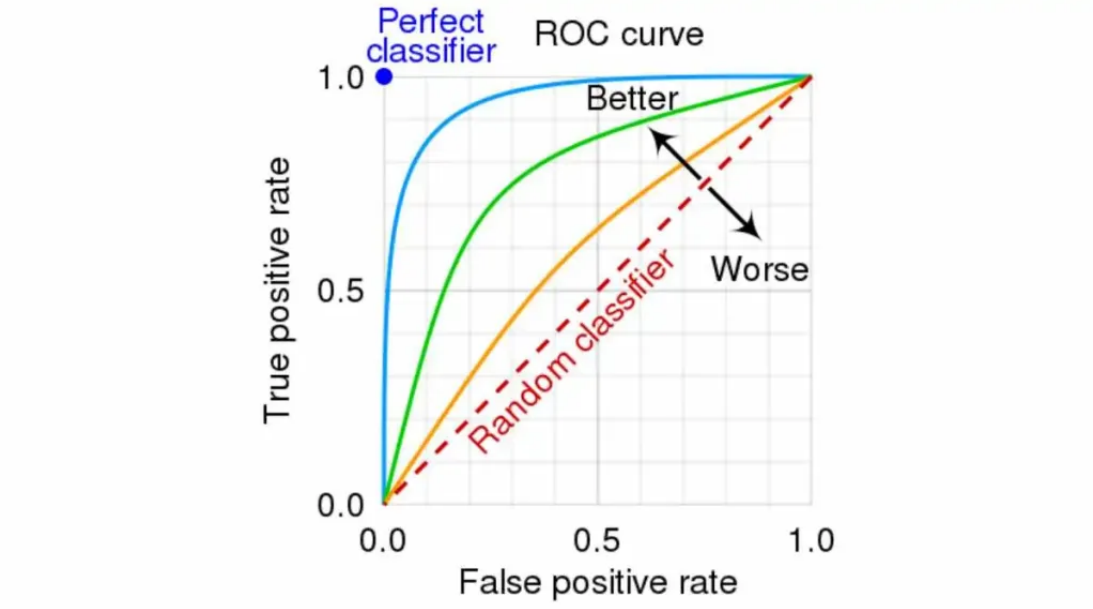
– S^+ and S^-: edges and non-edges,
– $p(u, v)$: the probability of an edge uv to be present in the geometric Chung-Lu model $\mathcal{G}(\mathbf{w}, \mathcal{E}, \alpha)$,
– build a binary classifier to see if $p(u, v)$ can be used to distinguish edges from non-edges.

Local Score

– The ROC (Receiver Operating Characteristic) is a curve showing the performance of a classification model at all classification thresholds.

y-axis: True Positive Rate (Sensitivity) — the proportion of actual positive instances the model correctly identifies

x-axis: False Positive Rate (1-Specificity) — the proportion of actual negative instances incorrectly classified as positive



Local Score

– The AUC (area under the ROC curve) provides an aggregate measure of performance across all possible classification thresholds which can be interpreted as the probability that a randomly chosen positive sample is ranked higher that a negative sample:

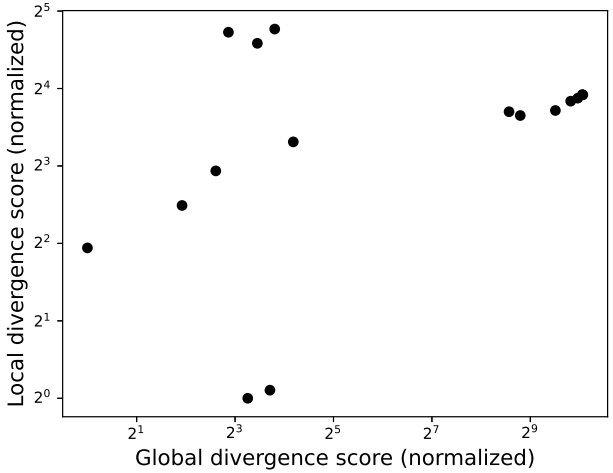
Local Score

– The AUC (area under the ROC curve) provides an aggregate measure of performance across all possible classification thresholds which can be interpreted as the probability that a randomly chosen positive sample is ranked higher that a negative sample:

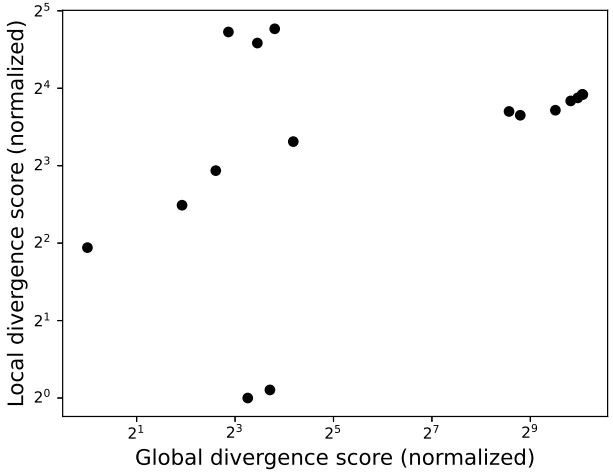
$$\frac{\sum_{st \in S^+} \sum_{uv \in S^-} 1\{p(s,t) > p(u,v)\}}{|S^+| \cdot |S^-|}.$$

(Can be easily approximated by sampling.)

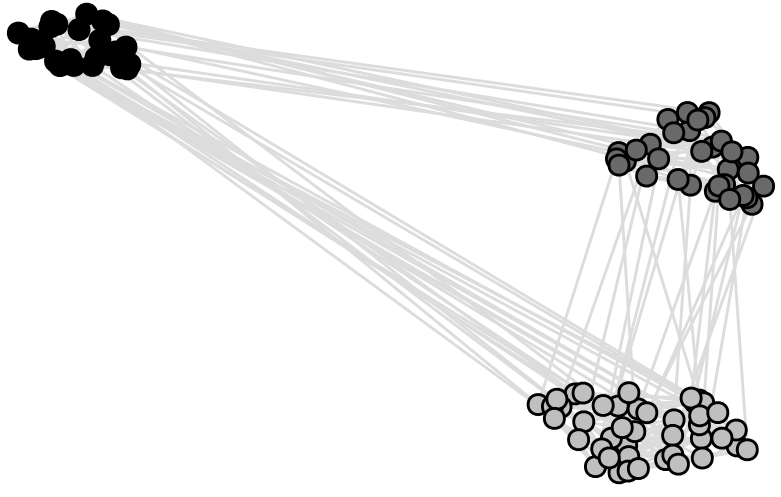
Example: ABCD (3 communities)



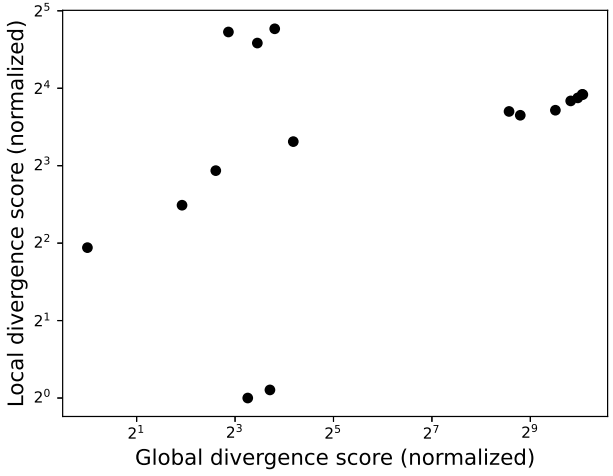
Example: ABCD (3 communities)



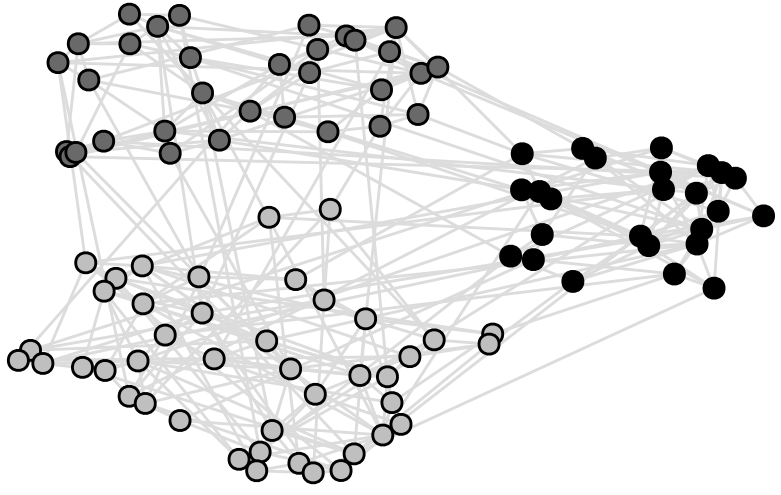
Low global divergence



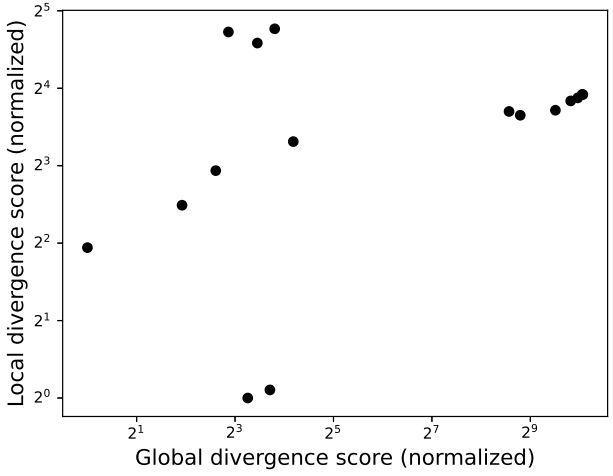
Example: ABCD (3 communities)



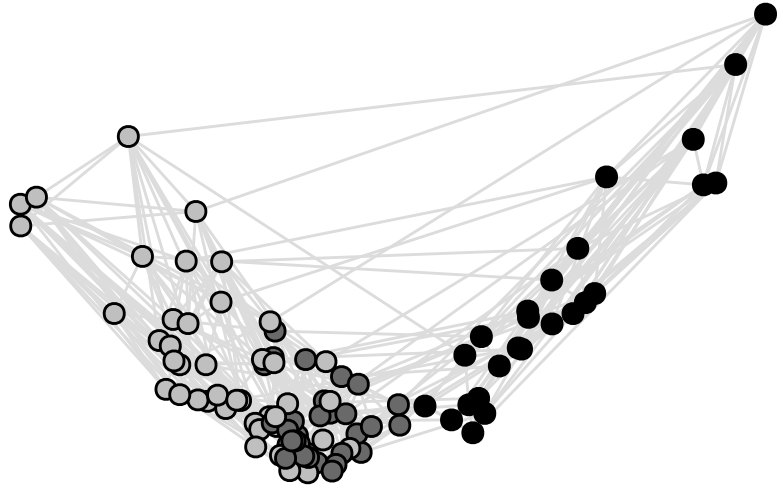
Low local divergence



Example: ABCD (3 communities)



Both divergences high



Geometric Chung-Lu model

We generalized the Chung-Lu model:

$$p_{i,j} = x_i x_j g(d_{i,j}),$$

where $d_{i,j} = \text{dist}(\mathscr{E}(v_i), \mathscr{E}(v_j))$ and $g(x) = g_\alpha(x)$ is some decreasing (similarity) function such as

$$g(d) = \left(1 - \frac{d - d_{\min}}{d_{\max} - d_{\min}}\right)^\alpha,$$

where $d_{\min}$ and $d_{\max}$ are the minimum, and respectively the maximum, distance between nodes in embedding $\mathscr{E}$.

Geometric Chung-Lu model

x_i's have to be tuned so that $\mathbb{E}[\deg(v_i)] = w_i$.

This part requires $\Theta(n^2)$ steps and so it is not feasible for large graphs.

Geometric Chung-Lu model

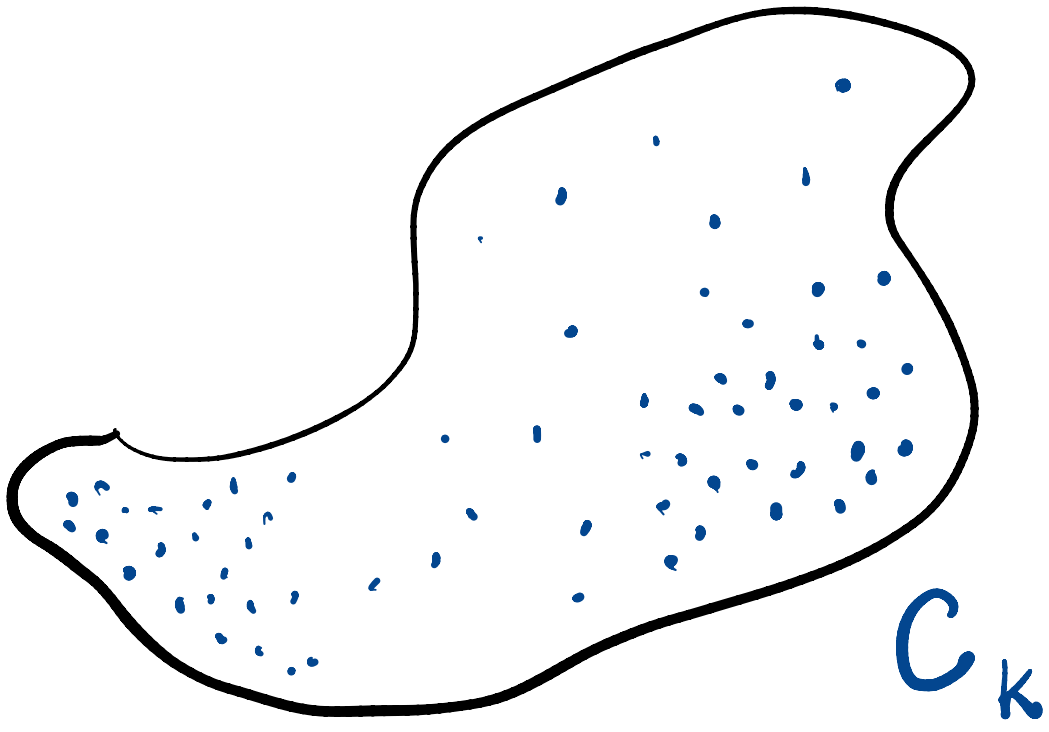
x_i's have to be tuned so that $\mathbb{E}[\deg(v_i)] = w_i$.

This part requires $\Theta(n^2)$ steps and so it is not feasible for large graphs.

We propose scalable (approximation) algorithm with running time $O(n \ln n)$.

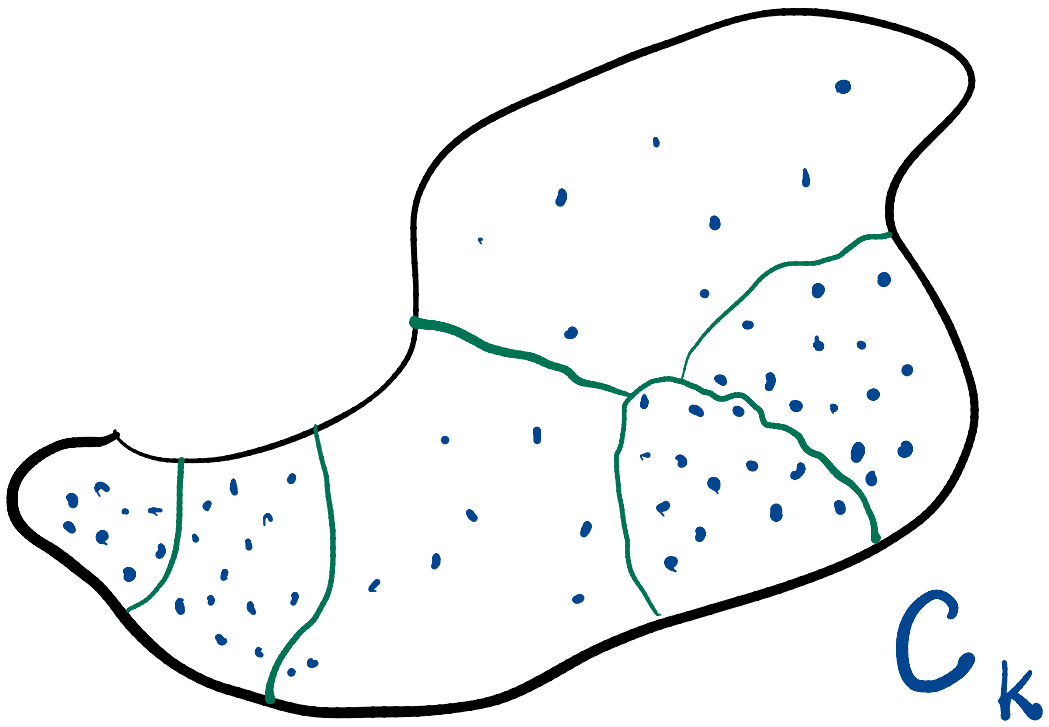
Scalable Algorithm — Main Idea

Independently consider each part.



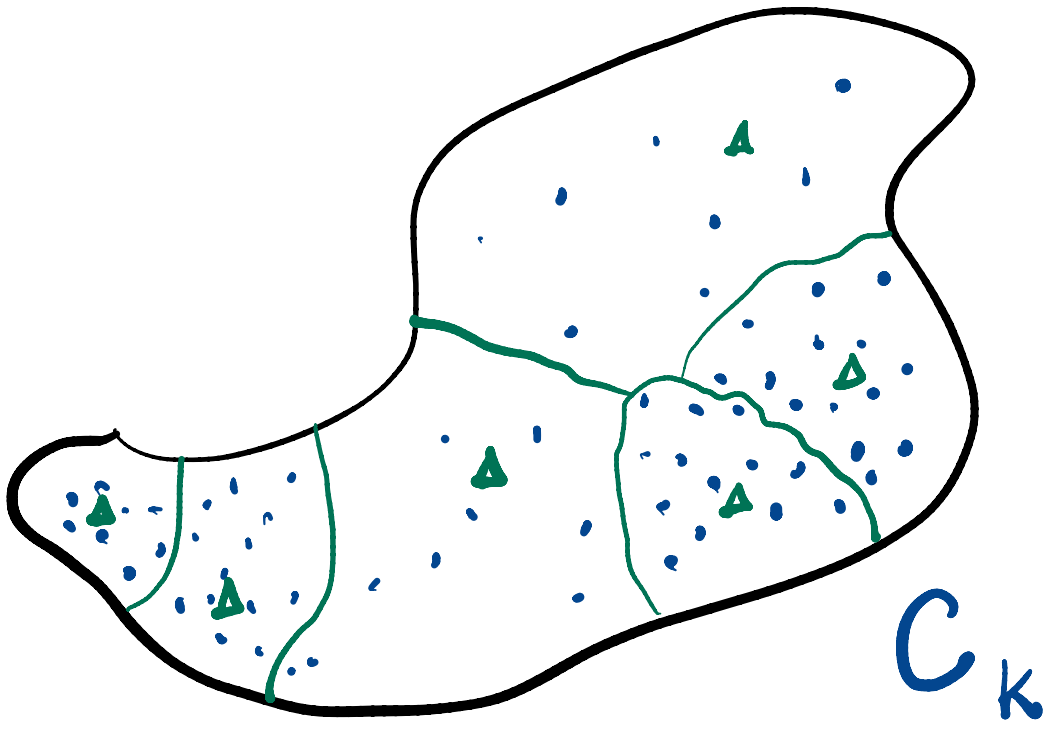
Scalable Algorithm — Main Idea

Group together nodes that are close to each other in the embedded space.



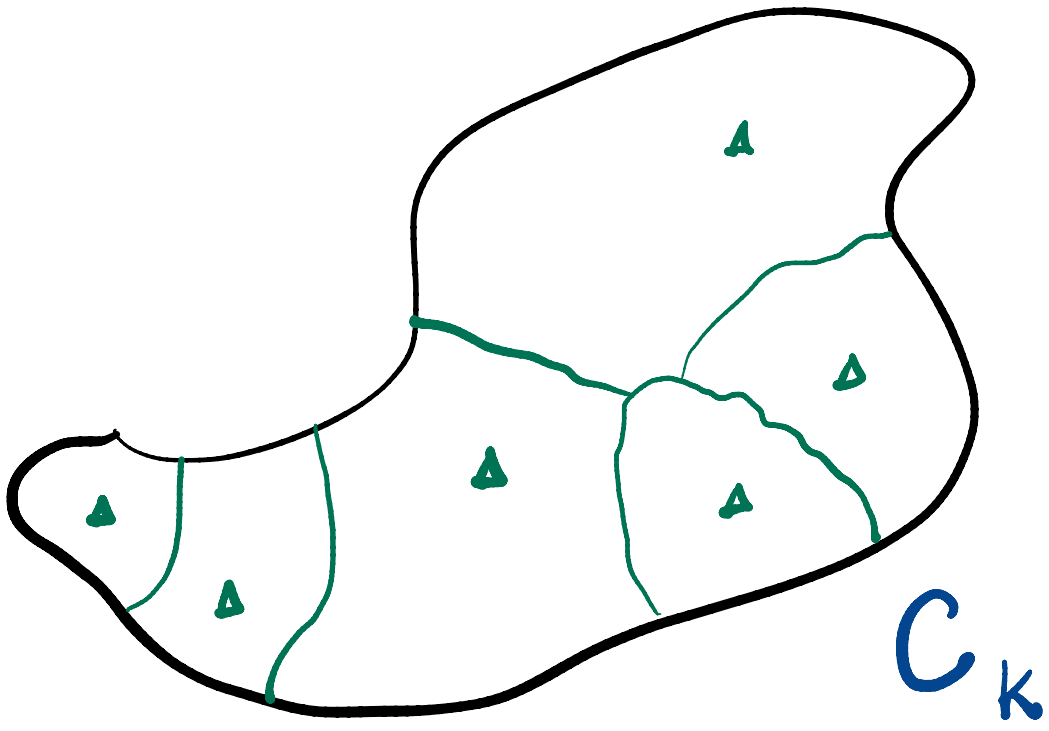
Scalable Algorithm — Main Idea

Replace each group by the corresponding auxiliary node (landmark) that is placed in the (appropriately weighted) center of mass.



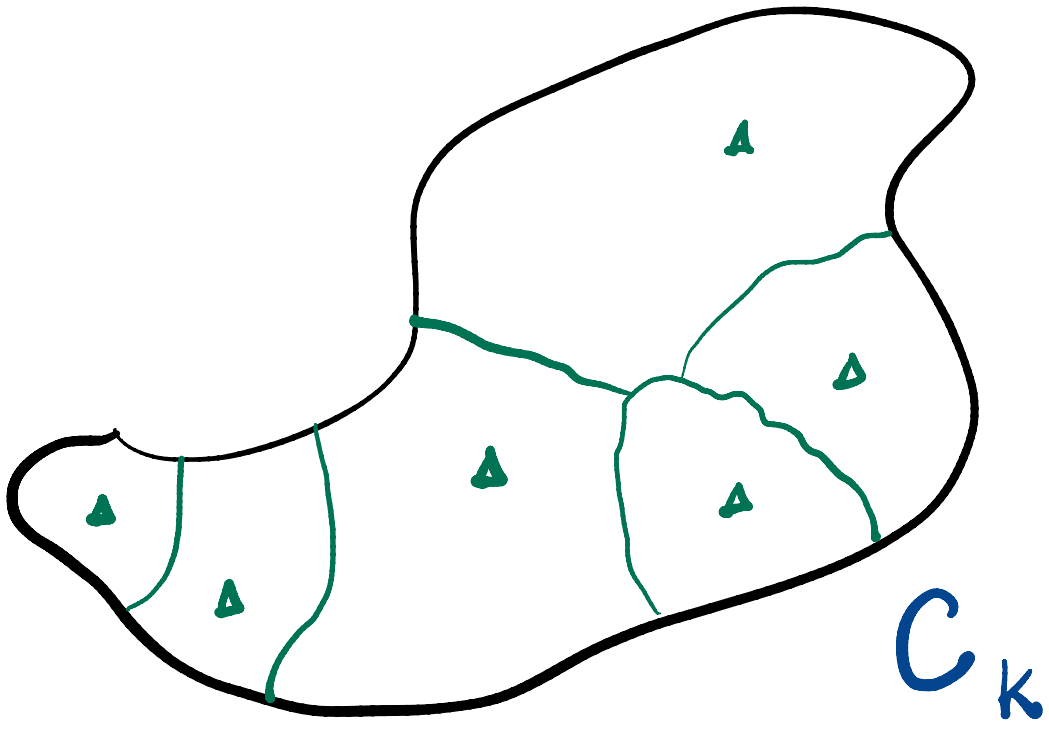
Scalable Algorithm — Main Idea

We get auxiliary graph with auxiliary embedding.
Approximate original $b_{i,j}$ by $a_{i,j}$ obtained for landmarks.



Scalable Algorithm — Main Idea

If the number of landmarks is $n' \approx \sqrt{n \ln n}$, then tuning weights takes $O((n')^2) = O(n \ln n)$.



Scalable Algorithm — Refining the partition

For each part we compute the weighted center of mass p_i and the weighted sum of squared errors (SSE, a natural measure of variation within a part) e_i:

$$p_i := \frac{\sum_{j \in C_i} w_j \, \mathscr{E}(v_j)}{\sum_{j \in C_i} w_j} \qquad \text{and} \qquad e_i := \sum_{j \in C_i} w_j \, \operatorname{dist}\big(p_i, \mathscr{E}(v_j)\big)^2.$$

Scalable Algorithm — Refining the partition

For each part we compute the weighted center of mass p_i and the weighted sum of squared errors (SSE, a natural measure of variation within a part) e_i:

$$p_i := \frac{\sum_{j \in C_i} w_j \, \mathscr{E}(v_j)}{\sum_{j \in C_i} w_j} \qquad \text{and} \qquad e_i := \sum_{j \in C_i} w_j \, \mathrm{dist}\big(p_i, \mathscr{E}(v_j)\big)^2.$$

Split each part a few times, and then keep splitting greedily (parts with the largest SSE).

Scalable Algorithm — Refining the partition

For each part we compute the weighted center of mass p_i and the weighted sum of squared errors (SSE, a natural measure of variation within a part) e_i:

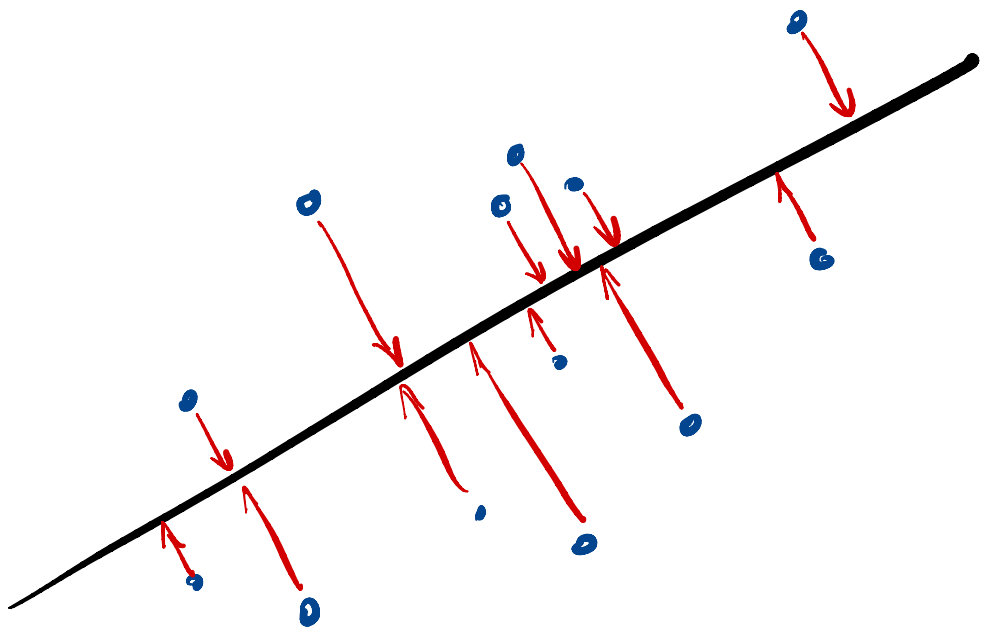
$$p_i := \frac{\sum_{j \in C_i} w_j \, \mathscr{E}(v_j)}{\sum_{j \in C_i} w_j} \qquad \text{and} \qquad e_i := \sum_{j \in C_i} w_j \, \operatorname{dist}\big(p_i, \mathscr{E}(v_j)\big)^2.$$

Split each part a few times, and then keep splitting greedily (parts with the largest SSE).

Would be nice to split a part with SSE equal to e_i into two parts such that $\max\{e_i^1, e_1^2\}$ is as small as possible—difficult and computationally expensive.

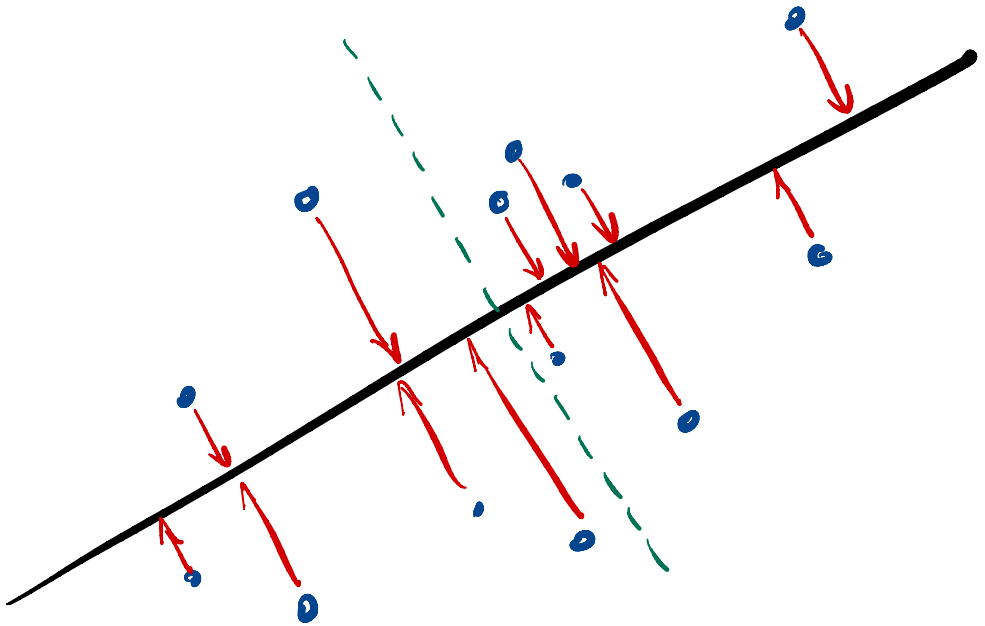
Scalable Algorithm — Refining the partition

Approximate using the well-known weighted Principal Component Analysis (PCA) in which the first principal component has the largest possible weighted variance.



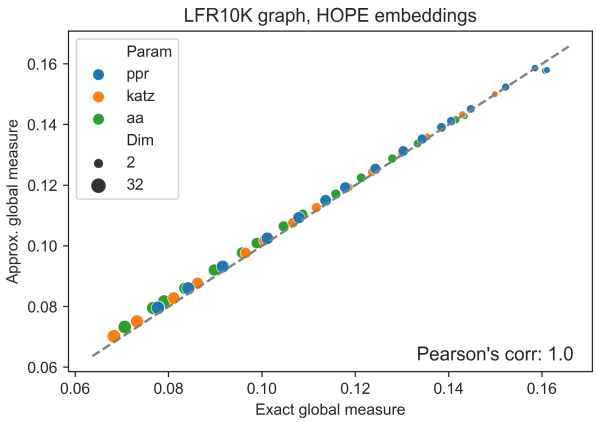
Scalable Algorithm — Refining the partition

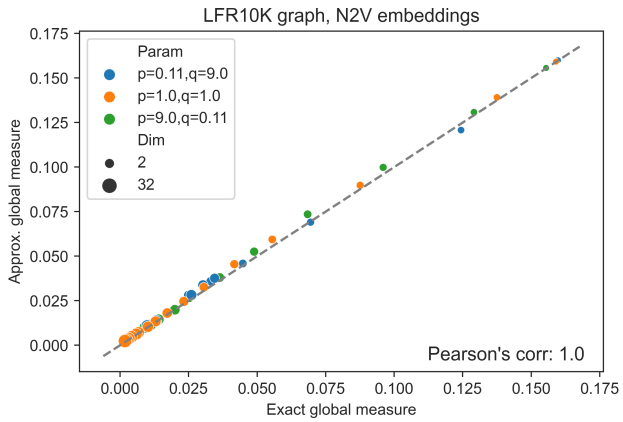
We get a total order of these points and one can quickly check which of the natural $|C_k| - 1$ partitions minimizes $\max\{e_i^1, e_1^2\}$.



Scalable Algorithm — Quality and Speed Comparison

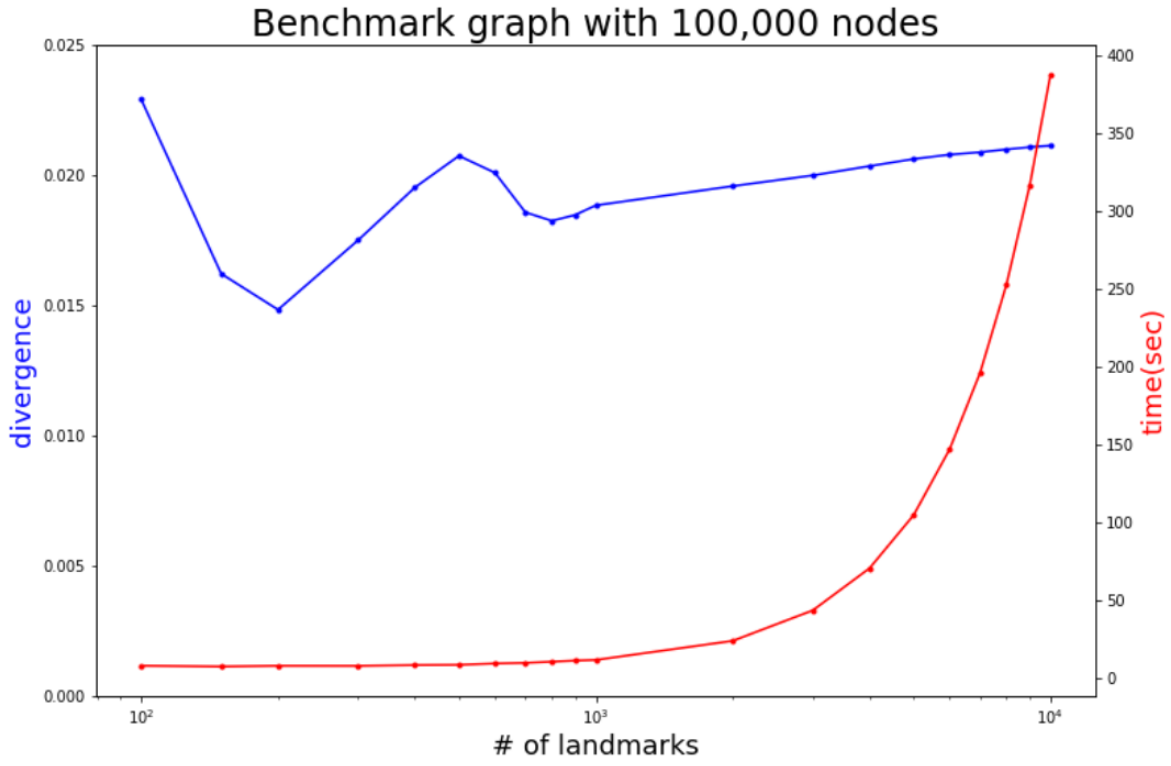
LFR graph (HOPE and Node2Vec embeddings)





Scalable Algorithm — Quality and Speed Comparison

ABCD graph



We tested n' up to $n' = 10^4 = n^{4/5}$, larger than $n' = \sqrt{n}$.
One can easily deal with graphs or order $n = (10^4)^2 = 10^8$.

A Few Applications

Datasets and Algorithms Used

– Zachary karate club data set (small, good for visualization),

Datasets and Algorithms Used

– Zachary karate club data set (small, good for visualization),

– synthetic ABCD graph on $n = 1{,}000$ nodes partitioned into 12 communities (sizes ranging between 50 and 150), $m = 8{,}327$ edges, and relatively weak community association ($\xi = 0.6$).

Datasets and Algorithms Used

– Zachary karate club data set (small, good for visualization),

– synthetic ABCD graph on $n = 1{,}000$ nodes partitioned into 12 communities (sizes ranging between 50 and 150), $m = 8{,}327$ edges, and relatively weak community association ($\xi = 0.6$).

– Embedding in 64-dimensional space using HOPE together with Personalized PageRank proximity measure. (It gives good divergence results when using the framework but other choices give similar results).

Node Classification

Embedding algorithms can be viewed as the process of extracting features of the nodes based on the structure of the graph, one may reduce the problem to classical machine learning predictive modeling classification problem for the set of vectors.

Node Classification

Embedding algorithms can be viewed as the process of extracting features of the nodes based on the structure of the graph, one may reduce the problem to classical machine learning predictive modeling classification problem for the set of vectors.

There are many algorithms, such as logistic regression, k-nearest neighbours, decision trees, support vector machine, etc., for any potential scenario that one might be interested in, including binary, multi-class, and multi-label classifications.

Node Classification

- Start with the synthetic ABCD graph (12 communities).

Node Classification

- Start with the synthetic ABCD graph (12 communities).
- Each node is represented by its 48-dimensional embedding, feature vector in machine learning terminology.

Node Classification

- Start with the synthetic ABCD graph (12 communities).
- Each node is represented by its 48-dimensional embedding, feature vector in machine learning terminology.
- Partition nodes randomly into a training set (25%) and a test set (75%).

Node Classification

- Start with the synthetic ABCD graph (12 communities).
- Each node is represented by its 48-dimensional embedding, feature vector in machine learning terminology.
- Partition nodes randomly into a training set (25%) and a test set (75%).
– We trained the random forest classifier (ensemble learning method for classification, regression, and other related tasks that operate by constructing a multitude of decision trees) on the training set.

Node Classification

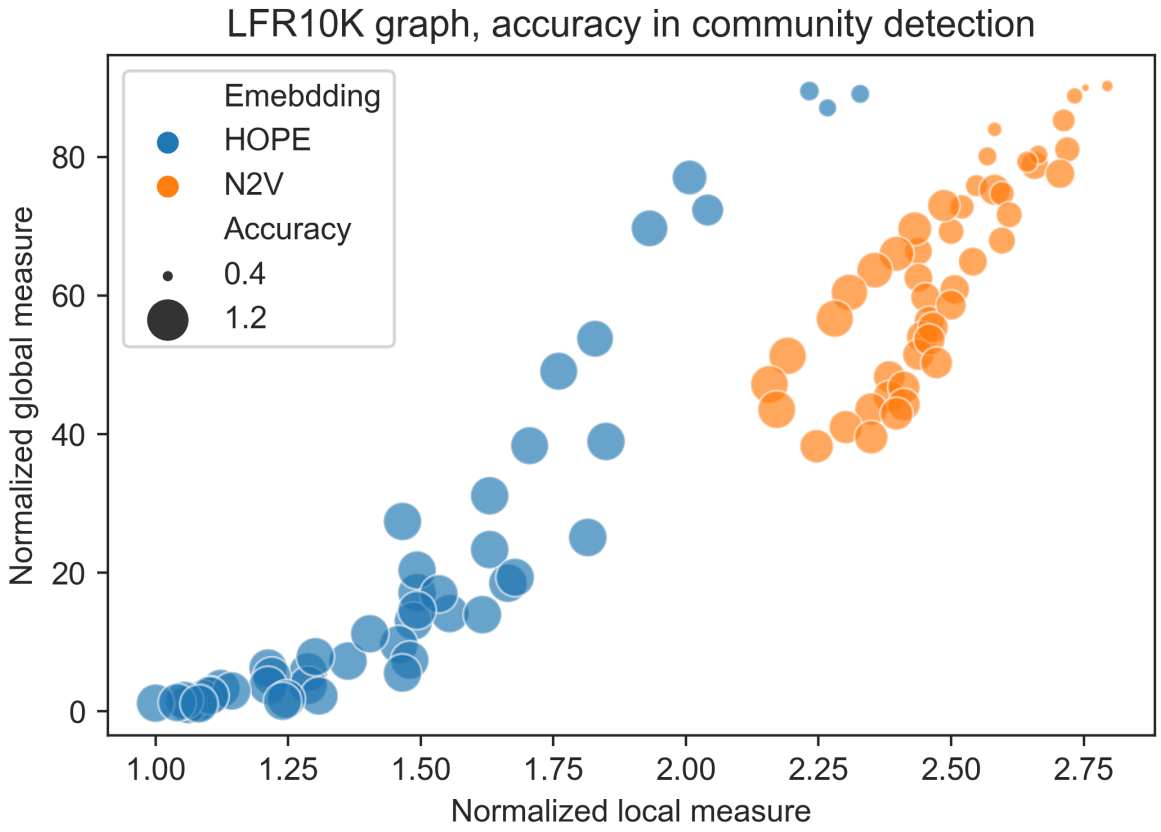
- Start with the synthetic ABCD graph (12 communities).
- Each node is represented by its 48-dimensional embedding, feature vector in machine learning terminology.
- Partition nodes randomly into a training set (25%) and a test set (75%).
– We trained the random forest classifier (ensemble learning method for classification, regression, and other related tasks that operate by constructing a multitude of decision trees) on the training set.
– We applied the model to the test set. We got 90.9% overall accuracy which is quite good in comparison to a random classifier which gives accuracy under 9%.

For multi-class classification, we often present the confusion matrix $\mathbf{C} = (c_{i,j})_{i,j\in[n]}$; $c_{i,j}$ is the number of nodes in community i classified as being in community j. The proportion of weight on the diagonal is the overall accuracy (again, 90.9%).

$$\mathbf{C} = \begin{bmatrix}
78 & 1 & 4 & 0 & 0 & 2 & 3 & 0 & 3 & 0 & 1 & 0 \\
0 & 77 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 71 & 0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\
1 & 1 & 1 & 64 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 2 & 1 & 77 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 54 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 3 & 53 & 3 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 42 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 45 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 44 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 42 & 0 \\
1 & 1 & 1 & 0 & 0 & 4 & 2 & 2 & 1 & 0 & 0 & 35
\end{bmatrix}$$

Node Classification

Correlation between the global and local scores and the quality of node classification.



Community Detection

– Embed nodes in ℓ-dimensional space.

– Ignore the initial graph and apply some generic clustering (scalable) algorithm to the set of associated vectors, such as k-means, DBSCAN, or HDBSCAN.

Community Detection

– Embed nodes in ℓ-dimensional space.
– Ignore the initial graph and apply some generic clustering (scalable) algorithm to the set of associated vectors, such as k-means, DBSCAN, or HDBSCAN.

One may want to combine the two clustering techniques which presumably should give better results as node embeddings provide some additional information about a function or a role of particular nodes, something that is not available with graph clustering alone.

We try something simple: k-Leiden.

Community Detection

– Use the same ABCD graph (12 communities) and the embedding as before.

Community Detection

– Use the same ABCD graph (12 communities) and the embedding as before.
– Compare the results of two graph clustering algorithms (Louvain and ECG), and two clustering algorithms using the embedding as feature vectors (k-means and DB-scan).

Community Detection

– Use the same ABCD graph (12 communities) and the embedding as before.
– Compare the results of two graph clustering algorithms (Louvain and ECG), and two clustering algorithms using the embedding as feature vectors (k-means and DB-scan).
– For k-means, we used $k \in \{6, 9, 12, 15, 24\}$).

Community Detection

– Use the same ABCD graph (12 communities) and the embedding as before.
– Compare the results of two graph clustering algorithms (Louvain and ECG), and two clustering algorithms using the embedding as feature vectors (k-means and DB-scan).
– For k-means, we used $k \in \{6, 9, 12, 15, 24\}$).
– Compare the results with the ground-truth using the Adjusted Mutual Information (AMI).

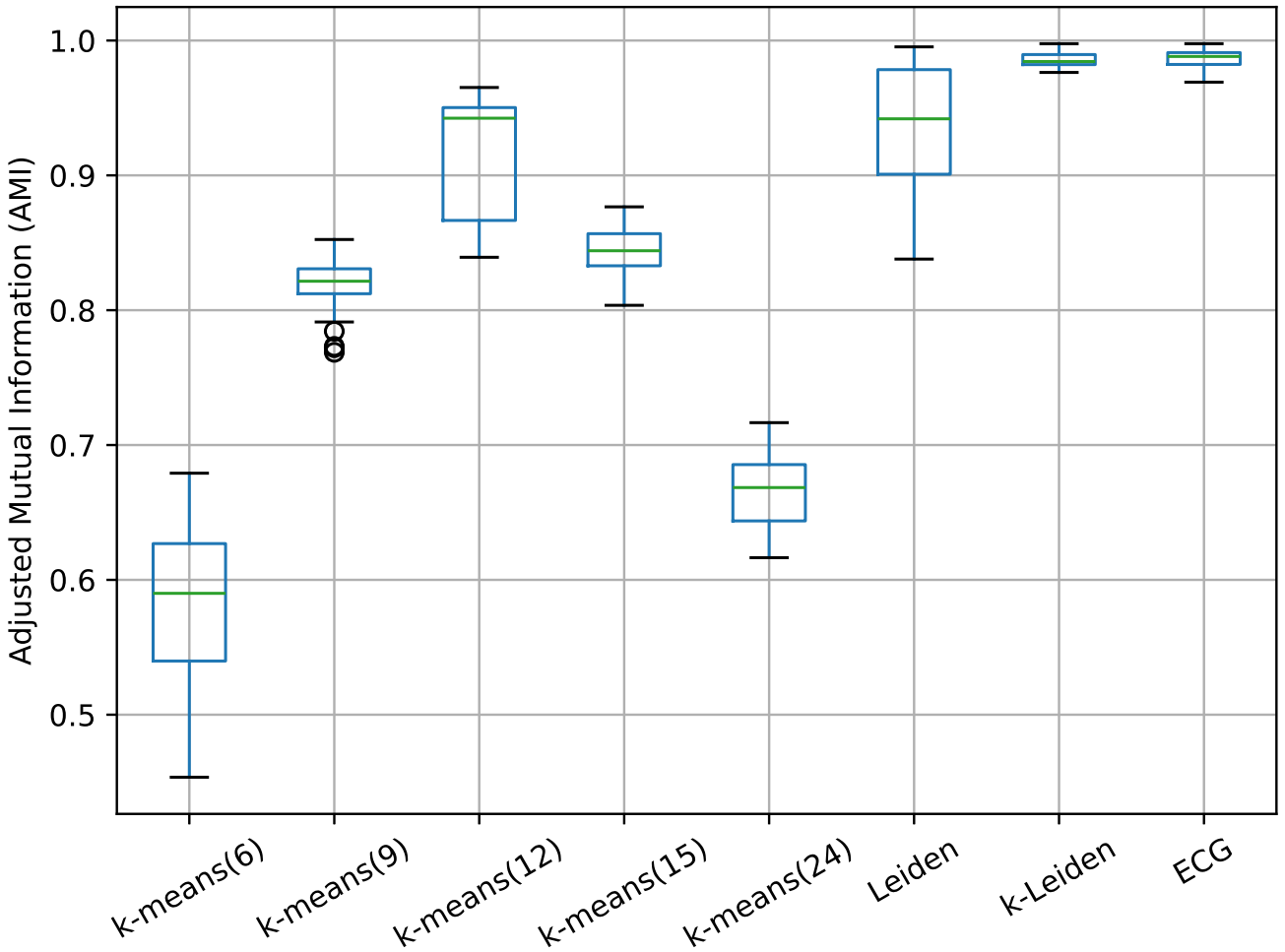
Community Detection

– Use the same ABCD graph (12 communities) and the embedding as before.
– Compare the results of two graph clustering algorithms (Louvain and ECG), and two clustering algorithms using the embedding as feature vectors (k-means and DB-scan).
– For k-means, we used $k \in \{6, 9, 12, 15, 24\}$).
– Compare the results with the ground-truth using the Adjusted Mutual Information (AMI).
– For k-means, good results are obtained in embedded space, provided that the correct number of clusters (12) is supplied.

Community Detection

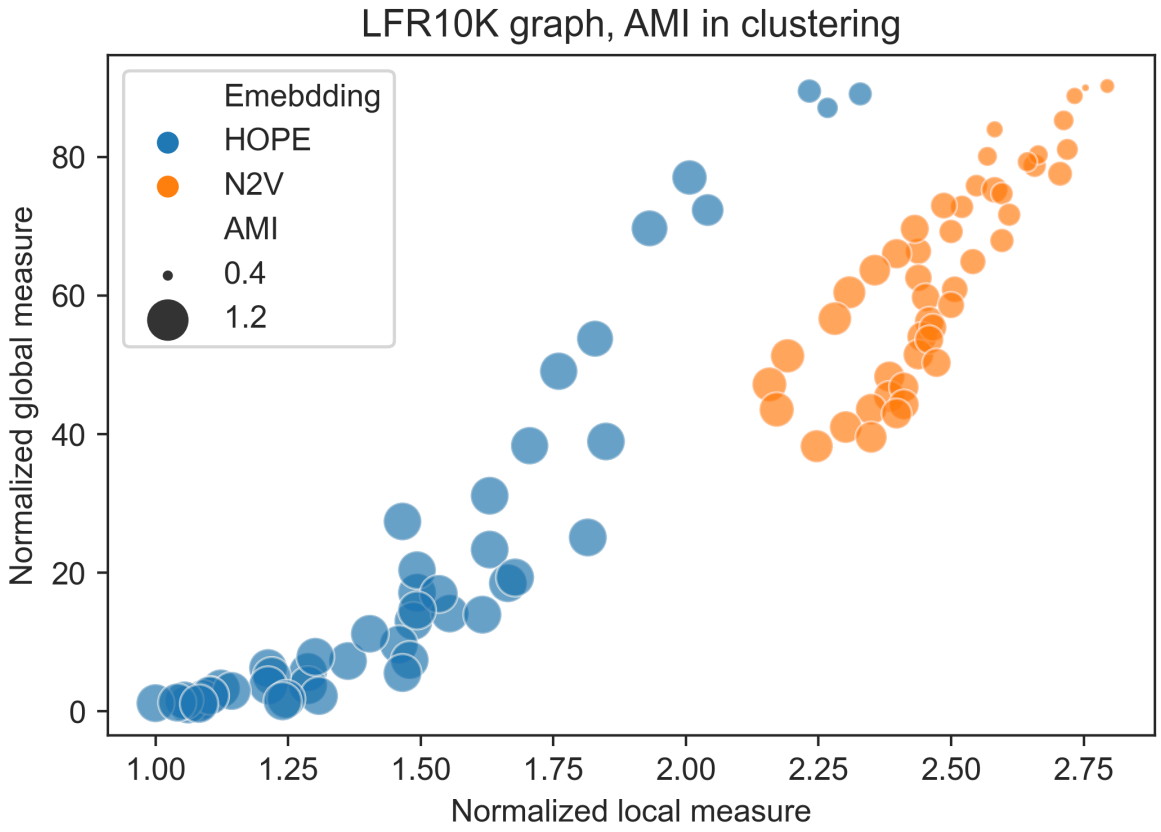
– Use the same ABCD graph (12 communities) and the embedding as before.
– Compare the results of two graph clustering algorithms (Louvain and ECG), and two clustering algorithms using the embedding as feature vectors (k-means and DB-scan).
– For k-means, we used $k \in \{6, 9, 12, 15, 24\}$).
– Compare the results with the ground-truth using the Adjusted Mutual Information (AMI).
– For k-means, good results are obtained in embedded space, provided that the correct number of clusters (12) is supplied.
– For DB-scan, since this algorithm is deterministic, we only report that the AMI value is equal to 0.67 for all nodes, and 0.88 if outliers are identified by the algorithm and then excluded.

Community Detection



Community Detection

Correlation between the global and local scores and the quality of node classification.



Link Prediction and Missing Links

Node embeddings can also be used to predict:
– missing links (important in biological networks; expensive/difficult/not accurate),
– links that are likely to be formed in the future (main ingredient of recommender systems).

Link Prediction and Missing Links

Node embeddings can also be used to predict:
– missing links (important in biological networks; expensive/difficult/not accurate),
– links that are likely to be formed in the future (main ingredient of recommender systems).

– Nodes that are close to each other in the embedded space but are not adjacent might get connected in the near future.
– Nodes that are far in the embedded space might get disconnected.

Such behaviour is known in social science as homophily and aversion.

Link Prediction and Missing Links

– Use the same ABCD graph G.
– Remove randomly 10% of edges to form G'.

Link Prediction and Missing Links

– Use the same ABCD graph G.
– Remove randomly 10% of edges to form G'.
– The positive class = all pairs of adjacent nodes.
– The negative class = random pairs of non-adjacent nodes.
– Both classes have $m' = |E(G')|$ pairs; training set is balanced.

Link Prediction and Missing Links

– Use the same ABCD graph G.
– Remove randomly 10% of edges to form G'.
– The positive class = all pairs of adjacent nodes.
– The negative class = random pairs of non-adjacent nodes.
– Both classes have $m' = |E(G')|$ pairs; training set is balanced.
– Compute the embedding $\mathscr{E}'$ using graph G' and divide the data into training and validation sets to test various binary operators that combine embeddings of two nodes into a feature vector to be used for prediction.

Link Prediction and Missing Links

– Use the same ABCD graph G.
– Remove randomly 10% of edges to form G'.
– The positive class = all pairs of adjacent nodes.
– The negative class = random pairs of non-adjacent nodes.
– Both classes have $m' = |E(G')|$ pairs; training set is balanced.
– Compute the embedding $\mathscr{E}'$ using graph G' and divide the data into training and validation sets to test various binary operators that combine embeddings of two nodes into a feature vector to be used for prediction.
– We used the logistic regression model (using the L_1-distance between the nodes' embeddings) in which the output is an estimation of the probability for the positive class in the training data set.

Link Prediction and Missing Links

– We re-trained our model on all data from graph G', and applied it to the edges deleted from graph G as well as a random sample of pairs of non-adjacent nodes in G.

Link Prediction and Missing Links

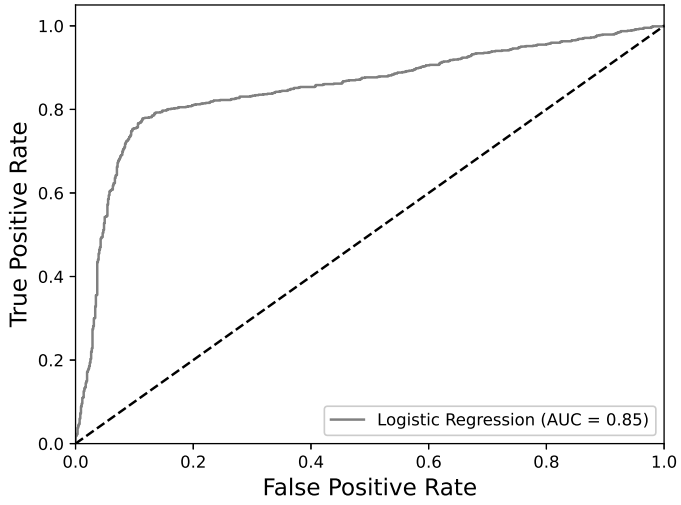
– We re-trained our model on all data from graph G', and applied it to the edges deleted from graph G as well as a random sample of pairs of non-adjacent nodes in G.
– We got 61% accuracy for $\xi = 0.6$ and 83% for $\xi - 0.2$ — it works well in comparison to the random classifier that has 50% accuracy.

Link Prediction and Missing Links

– We re-trained our model on all data from graph G', and applied it to the edges deleted from graph G as well as a random sample of pairs of non-adjacent nodes in G.

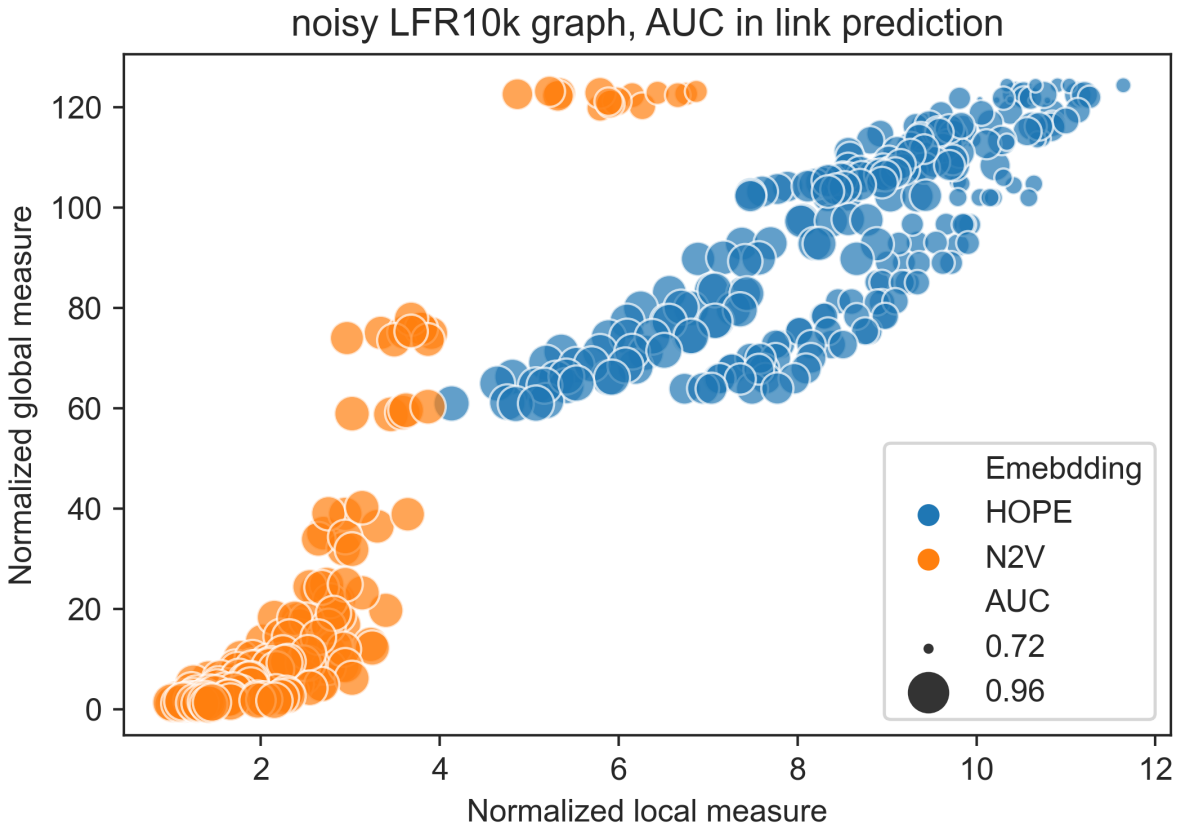
– We got 61% accuracy for $\xi = 0.6$ and 83% for $\xi - 0.2$ — it works well in comparison to the random classifier that has 50% accuracy.

– The results are usually presented in the form of the Receiver Operating Characteristic curve (ROC curve).



Link Prediction and Missing Links

Correlation between the global and local scores and the quality of link prediction.



Visualization

– There are some specialized algorithms (layouts).

Visualization

– There are some specialized algorithms (layouts).

– Embed G in k-dimensional space (k relatively large).
– Use some generic dimensionality reduction techniques to go down to 2D: Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbour Embedding (t-SNE).

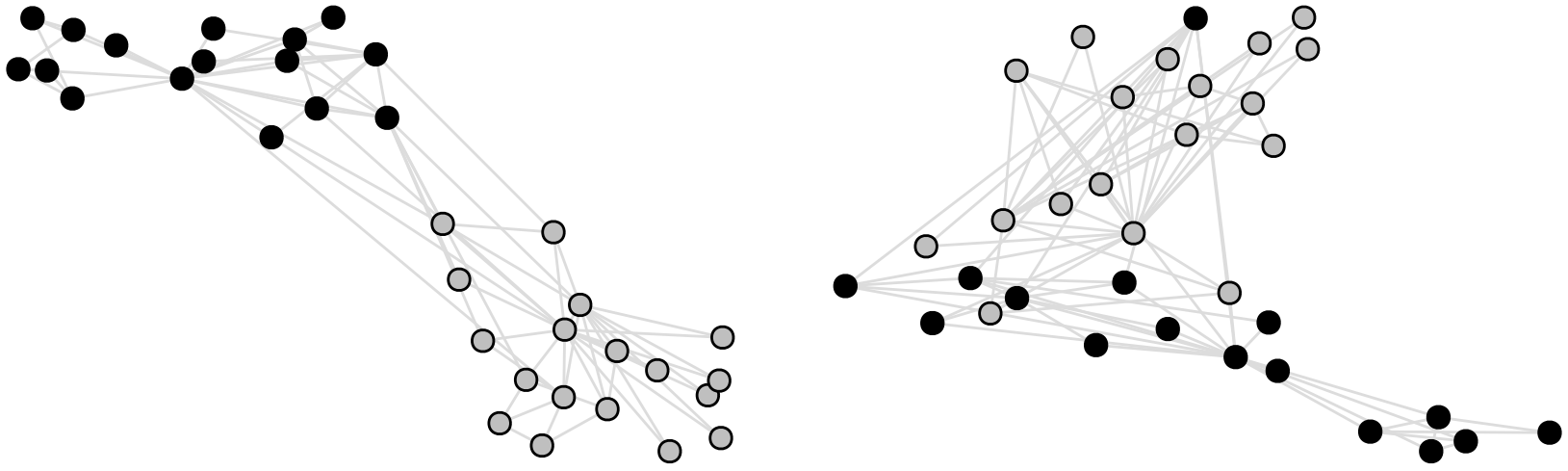
Visualization

– There are some specialized algorithms (layouts).

– Embed G in k-dimensional space (k relatively large).
– Use some generic dimensionality reduction techniques to go down to 2D: Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbour Embedding (t-SNE).

– We personally recommend Uniform Manifold Approximation and Projection (UMAP) that is a novel manifold learning technique for dimension reduction.
– We also recommend to test various embeddings and various parameters and select the one that scores well in the benchmark framework.

Visualization



Other Directions

Signed Graphs: each edge has a positive or negative sign (friendly or antagonistic interactions). There are some embeddings, such as Signed Network Embedding (SNE), that are crafted to deal with such networks.

Other Directions

Signed Graphs: each edge has a positive or negative sign (friendly or antagonistic interactions). There are some embeddings, such as Signed Network Embedding (SNE), that are crafted to deal with such networks.

Embedding Edges: represent edges as low-dimensional vectors. One can embed the nodes of the line graph but there are more approaches.

Other Directions

Signed Graphs: each edge has a positive or negative sign (friendly or antagonistic interactions). There are some embeddings, such as Signed Network Embedding (SNE), that are crafted to deal with such networks.

Embedding Edges: represent edges as low-dimensional vectors. One can embed the nodes of the line graph but there are more approaches.

Multi-Layered Graphs: sometimes we have ℓ graphs available with overlapping sets of nodes. For example, in protein-protein interaction networks derived from different tissues (say, from brain and liver tissues), some proteins occur across multiple tissues.

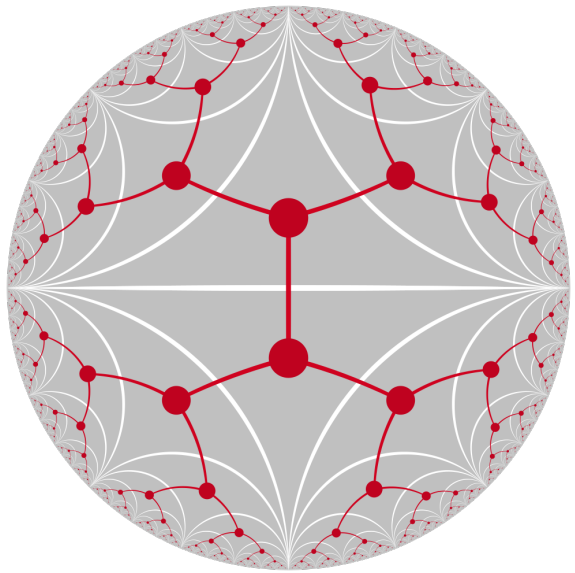
Other Directions

Hyperbolic Spaces: real-world graphs with scale-free or hierarchical structure might produce outcomes with relatively large distortion (graph distances cannot be accurately estimated based on the Euclidean distances).

Other Directions

Hyperbolic Spaces: real-world graphs with scale-free or hierarchical structure might produce outcomes with relatively large distortion (graph distances cannot be accurately estimated based on the Euclidean distances).

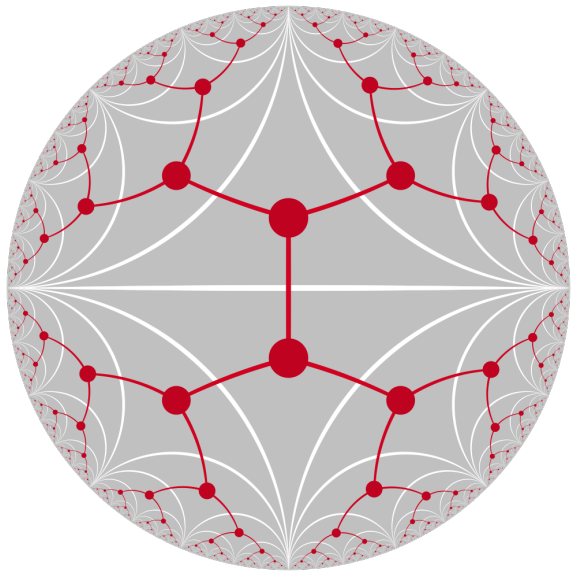
Hyperbolic geometries might help!



Other Directions

Hyperbolic Spaces: real-world graphs with scale-free or hierarchical structure might produce outcomes with relatively large distortion (graph distances cannot be accurately estimated based on the Euclidean distances).

Hyperbolic geometries might help!



Hyperbolic Graph Convolutional Neural Network (HGCN) is the first inductive hyperbolic counterpart of GCN.

Other Directions

Embedding Graphs: map the graph to a point in a vector space.

Two families: explicit graph embeddings (return embeddings) and implicit ones that are also known as graph kernels (provide a pairwise similarity measure between graphs).

Other Directions

Embedding Graphs: map the graph to a point in a vector space.

Two families: explicit graph embeddings (return embeddings) and implicit ones that are also known as graph kernels (provide a pairwise similarity measure between graphs).

We experimented with Graph2Vec (deep learning) in complementary material.

We used two well-known datasets (NCI1 and NCI109) of chemical compounds from the National Cancer Institute screened for activity against non-small cell lung cancer and ovarian cancer cell lines.

Experiments

Experiments

Jupyter notebooks and datasets that accompany the textbook/slides are available on GitHub (`Python_Notebooks_2nd` directory):

`https://github.com/ftheberge/GraphMiningNotebooks`

Tutorials for the Python notebooks can be found on YouTube:

`https://www.youtube.com/@MiningComplexNetworks`

THE END