Graph Neural Networks

Lecture 2

Danila Biktimirov
Applied Computer Science
Neapolis University Pafos

14 February 2025

Overview

Previously on...

Previously on...

Graphs as a Tool for Data Analysis

+ Represent many real-world systems
+ Help work with non-sequential data

- Very complex structure
- Require careful consideration of how to represent data

Previously on...

Types of Tasks on Graphs

1. Node Level
2. Edge Level
3. Graph Level

Manual Feature Engineering

Before Deep Learning

Before moving on to core deep learning methods, we should first explore existing approaches for creating so-called **hand-crafted features**.

By analogy with the types of existing tasks for graph data analysis, we will examine features for **nodes, edges, and graphs**.

IMPORTANT: We assume that all graphs discussed in the lecture will be **undirected**.

Formulating a Machine Learning Task on Graphs

Task: Make predictions for a set of objects.

Features:
Objects:
Loss function:

Problem Formulation

Task: Make predictions for a set of objects.

Features: n-dimensional vectors
Objects: Nodes, edges, graphs
Loss function: Depends on the task

Node-level

- Node degree
- Centrality
- Clustering coefficient
- Graphlets

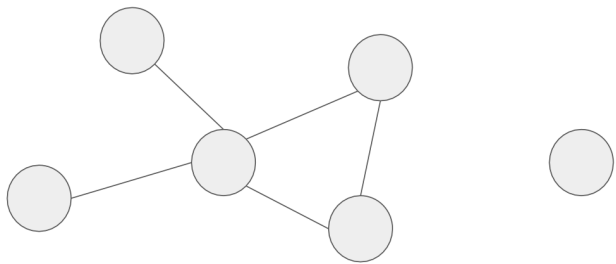
Node Degree

Definition: The degree of a node is the number of edges incident to it.

In the case of undirected graphs without loops, the node degree can be considered as the number of neighbors of that node.

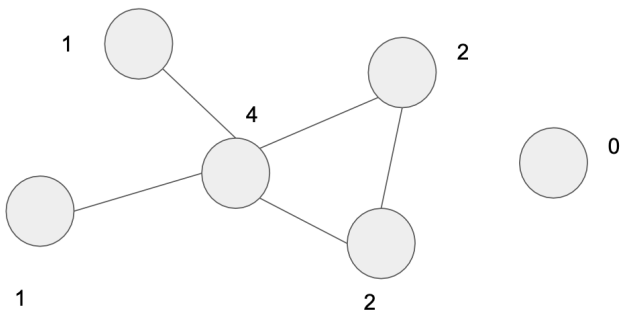
$$d_u = \sum_{v \in V} A[u, v]$$

Node Degree



Node Degree

The main property of this statistic is that all neighboring nodes are valued equally.



Graph-level Node Degree

The definition of node-level centrality can be extended to the entire graph, in which case we refer to **graph centralization**.

Let v^* be the node with the highest degree centrality in G. Let $X := (Y, Z)$ be a connected graph with $|Y|$ nodes that maximizes the following quantity (with y^* as the node with the highest degree centrality in X):

$$H = \sum_{j=1}^{|Y|} [C_D(y^*) - C_D(y_j)]$$

Accordingly, the **degree centralization** of graph G is given by:

$$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v^*) - C_D(v_i)]}{H}$$

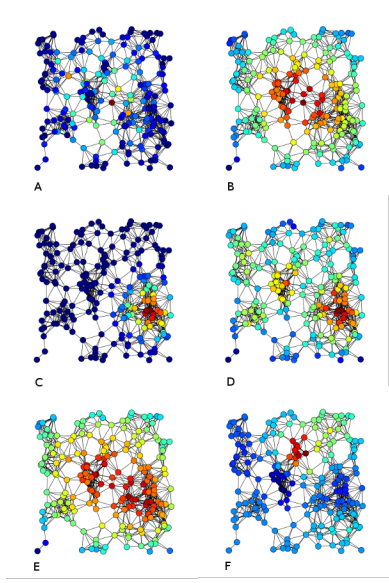
Graph-level Node Degree

$$H = (n-1) \cdot ((n-1)-1) = n^2 - 3n + 2.$$

Thus, for any graph $G := (V, E)$, the degree centralization is given by:

$$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v^*) - C_D(v_i)]}{|V|^2 - 3|V| + 2}$$

Centrality



Centrality

The node degree does not allow us to understand the **importance** of a node in the graph context.

To better understand the significance of a node in a graph, we can look at its **centrality measure**.

There are different ways to evaluate this statistic:

- From the perspective of eigenvectors
- From the perspective of neighborhood
- From the perspective of closeness

Eigenvector Centrality

Definition: Centrality is defined as the sum of the centralities of all neighbors.

The definition is recursive—this is not very convenient, so an analytical approach is needed.

$$c_u = \frac{1}{\lambda} \sum_{v \in V} A[u, v] c_v \quad \forall u \in V;$$

Eigenvector Centrality

Suppose $n = 3$ and

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Eigenvector Centrality

Then eigenvector centrality (with the normalization $\sum_{i \in N} c_i = 1$) is defined as the solution to the system of equations:

$$\lambda c_1 = c_2$$
$$\lambda c_2 = c_1$$
$$\lambda c_3 = c_1 + c_2$$
$$c_1 + c_2 + c_3 = 1.$$

Solving this system gives:

$$\lambda = 1, \quad c_1 = c_2 = \frac{1}{4}, \quad c_3 = \frac{1}{2}.$$

Eigenvector Centrality

If we define e as the vector of node centralities, we can transition to the standard eigenvalue equation for the adjacency matrix.

Thus, e is an eigenvector of the adjacency matrix A, where:

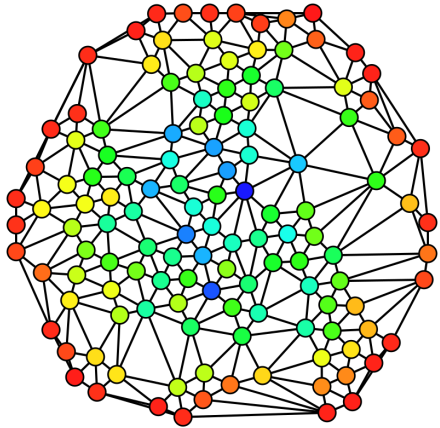
$$\lambda e = Ae$$

Betweenness Centrality

We can assume that a node v is important if it lies on a large number of shortest paths between other nodes that include this node.

$$c_v = \sum_{s \neq v \neq t} \frac{\text{number of shortest paths between } s \text{ and } t \text{ containing } v}{\text{number of shortest paths between } s \text{ and } t}$$

Betweenness Centrality



Closeness Centrality

We can assume that a node is important if it is very close to all other nodes.

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

Harmonic Centrality

In a (not necessarily connected) graph, **harmonic centrality** modifies the closeness centrality measure by applying summation and inversion:

$$H(x) = \sum_{y \neq x} \frac{1}{d(y, x)}$$

Clustering Coefficient

To identify communities, we can compute a statistic for a node that measures the connectivity among all of its neighbors.

$$c_u = \frac{|(v_1, v_2) \in \mathcal{E} : v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}}$$

From Clustering Coefficient to Graphlets

If we observe carefully, the clustering coefficient counts the number of *"triangles."*

This raises a natural question—what if we count not only triangles?

Graphlets

Graphlets are subgraphs that describe the structural network of a node's neighbors.

Essentially, they generalize previous feature description methods:

- The **node degree** represents the number of edges connected to a node.
- The **clustering coefficient** represents the number of triangles connected to a node.

Graphlets

There are 73 different graphlets of size 2–5 (while there are 30 non-isomorphic graphs).

Thus, the **graphlet vector** is a 73-dimensional vector that represents how many times a node is part of each specific graphlet. (Of course, the size is not limited to five.)

The meaning of **graphlet degree** is that it provides additional information about the **local topology**.

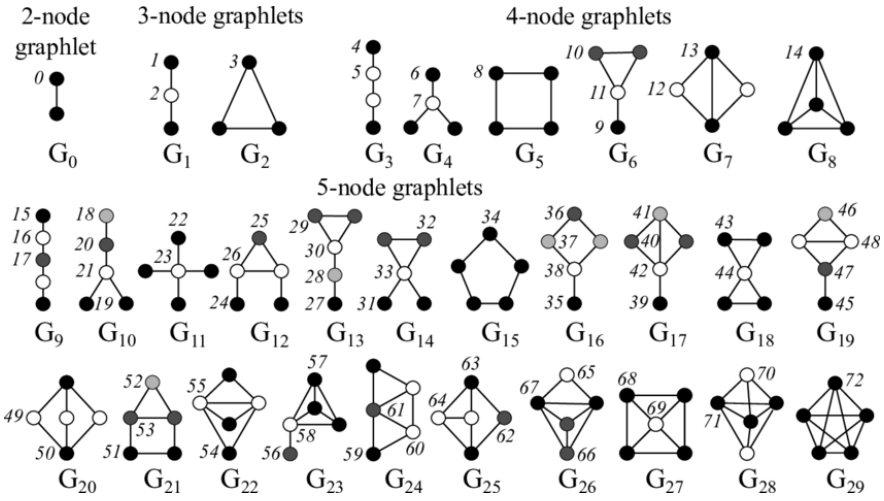
Graphlets

Definition: An **induced subgraph** is a subset of vertices and edges of a graph where the vertices are connected if they are connected in the original graph.

Definition: Isomorphic graphs—Two graphs are called isomorphic if they have the same number of vertices connected in the same way.

Definition: Graphlets—Connected, non-isomorphic induced subgraphs.

Graphlets


2-node graphlet
3-node graphlets
4-node graphlets
5-node graphlets
G0
G1
G2
G3
G4
G5
G6
G7
G8
G9
G10
G11
G12
G13
G14
G15
G16
G17
G18
G19
G20
G21
G22
G23
G24
G25
G26
G27
G28
G29

Predicting Missing Links

Given: A set of edges
Output: New edges

How can we design features for a pair of nodes?

Predicting Missing Links: Two Types of Problems

Random Edge Removal:

- Some nodes are removed from the graph.
- The goal is to predict their reappearance.

Edges Over Time:

- Given timestamps t_0 and t_0' for the graph.
- Generate a ranked list of nodes for timestamps t_1 and t_1'.

Predicting Missing Links

The main idea follows this approach:

- Compute a **score** for each pair of nodes in the graph.
- Rank the pairs.
- Predict the **top** n as new links.
- Check which links actually appeared in the final graph states at t_1 and t_2'.

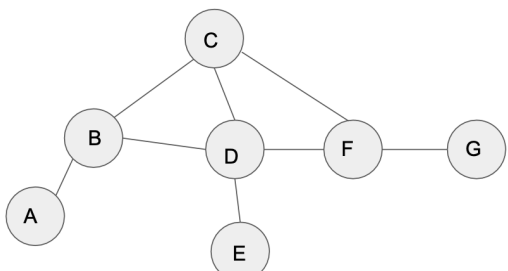
Predicting Missing Links: Features

We consider the following types of features:

- Distance
- Local overlap
- Global overlap

Distance

The distance between a pair of nodes can be defined as the length of the shortest path between them.



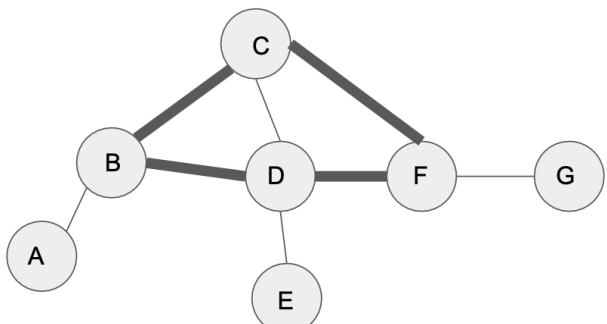
AC = CE = 2

BF = 2

BG = 3

Distance

The shortest paths BE and BF are equal, but from B to F, there exist two shortest paths. This provides additional information that can be extracted.



Local Overlap

It makes sense to analyze the sets of neighbors of two nodes.

Common Neighbors:

$$S[u, v] = |\mathcal{N}(u) \cap \mathcal{N}(v)|$$

Jaccard Coefficient:

$$S_{\text{Jaccard}}[u, v] = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}$$

Adamic-Adar Index:

$$S_{\text{AA}}[v_1, v_2] = \sum_{u \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)} \frac{1}{\log(d_u)}$$

Problems with Local Overlap

If two nodes do not share common neighbors, the selected local overlap metric will be zero.

This makes sense in the context of the current graph state, but as the graph evolves, these nodes might become connected over time. For a more comprehensive understanding, it is necessary to analyze the entire graph structure.

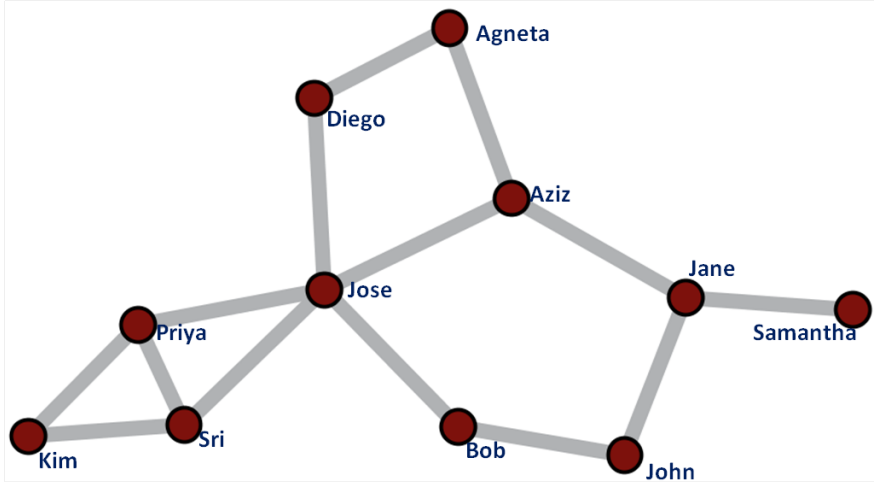
Global Overlap

What makes sense to analyze in an arbitrary graph if we want to evaluate the connection between two distant nodes?

A natural approach is to count all possible paths between two nodes. This leads to the **Katz Index**:

$$S_{\text{Katz}}[u, v] = \sum_{i=1}^{\infty} \beta^i A^i[u, v]$$

Katz Index



Katz Index

We can compute the **index matrix** (for all pairs of nodes at once):

$$S_{\text{Katz}} = \sum_{i=0}^{\infty} (I - \beta A)^{-1} - I$$

At the Graph Level

After analyzing features that characterize nodes and edges, it is time to explore how we can characterize an entire graph.

One option is to simply aggregate node statistics (which, in theory, is a good starting point).

Another approach is to use **graph kernels**. Once a kernel is defined, we can apply classical machine learning techniques, such as **kernel SVM**.

- **Graphlet Kernels**
- **Weisfeiler-Lehman Kernel**

Graphlet Kernels

The main idea is to borrow the concept of the "bag of words" model, replacing it with a **bag of nodes**.

To construct a **graphlet kernel**, we assume the following refinements:

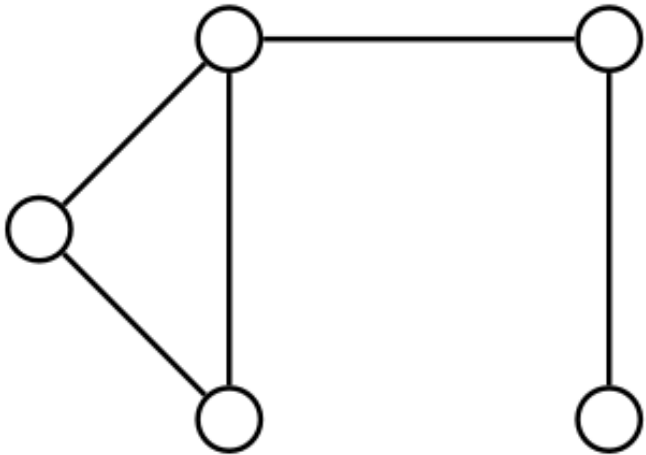
- Nodes in graphlets should not be connected.
- There is no root node.

Graphlet Kernels

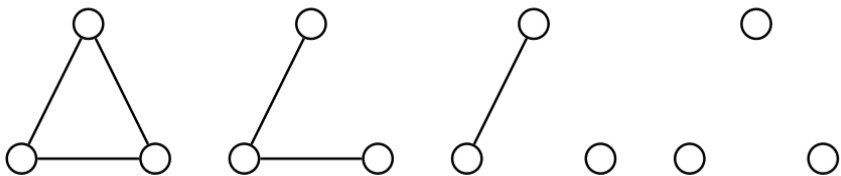
The **graphlet count vector** is a vector that represents the number of graphlets in a network.

$$K(G, G') = f_G^T f_{G'}$$

Graphlet Kernels



Graphlet Kernels



Graphlet Kernels

When comparing different graphs G and G', the result can be inconsistent.

Normalization is required:

$$h_G = \frac{f_G}{\text{Sum}(f_G)}$$

$$K(G, G') = h_G^T h_{G'}$$

Graphlet Kernels: Problems

Very expensive.

For a graph of size n and subgraph size k, the complexity is $**n^k**$.

Why? Because subgraph isomorphism is **NP-hard**.

If the degree of a node is bounded by d, the complexity reduces to $**nd^k**$.

Weisfeiler-Lehman Kernel

The core idea of the algorithm is **iterative neighborhood aggregation**.

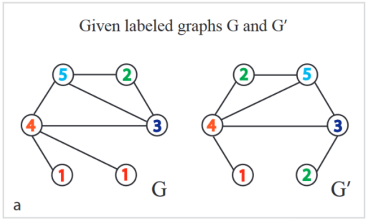
Algorithm:

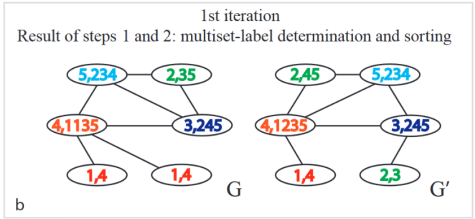
1. Assign a **label** to each node.
2. Update labels iteratively:

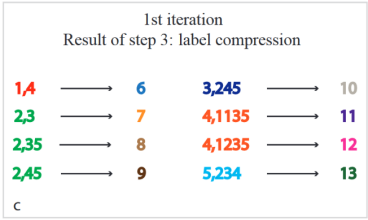
$$l^{(i)}(v) = \text{HASH}\left(\left\{l^{(i-1)}(u) \forall u \in \mathcal{N}(v)\right\}\right)$$

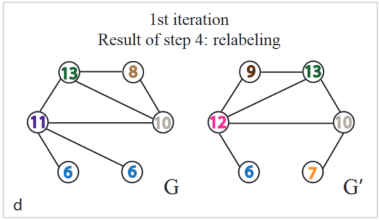
3. After k steps, this vector summarizes the structure of the k-**hop neighborhood**.

Weisfeiler-Lehman Kernel



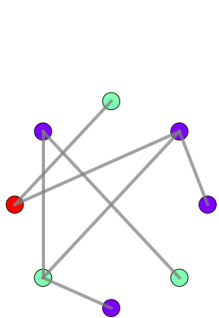




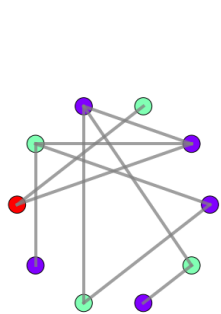


Weisfeiler-Lehman Kernel

Two labeled graphs

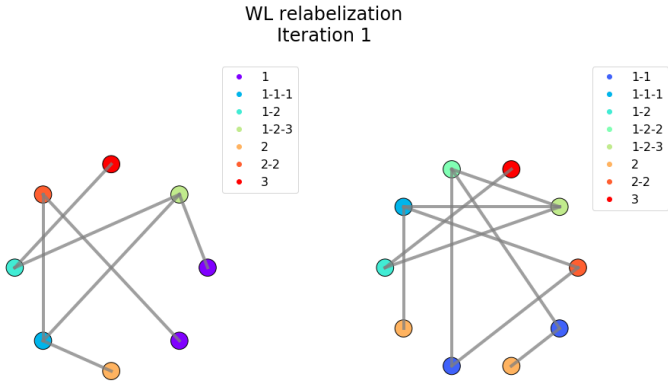


$\phi_0(G) = (4 \quad 3 \quad 1)$

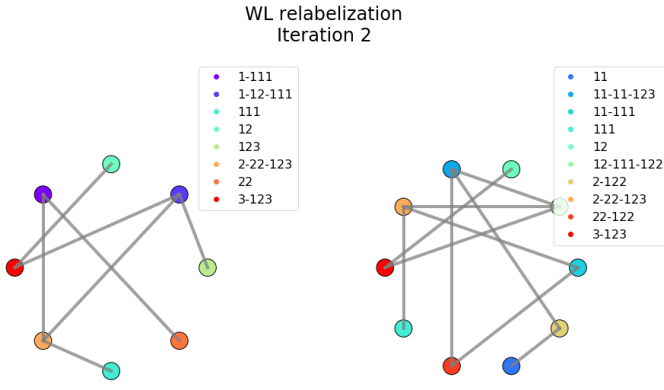


$\phi_0(G) = (5 \quad 4 \quad 1)$

Weisfeiler-Lehman Kernel



Weisfeiler-Lehman Kernel



$\phi_2(G) = (1\,1\,0\,0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,1)$ $\phi_2(G) = (0\,0\,1\,1\,1\,1\,1\,1\,0\,1\,1\,0\,1\,1)$

WL Kernel

- A **computationally efficient** method.
- When computing the kernel, it is sufficient to track **only the colors** appearing in both graphs.
- Counting colors forms a **sequence**.
- In general, this results in a **sequence over edges**.

The End?