Summary of the lecture „MA INF 4316 - Graph Representation Learning“ by Dr. Pascal Welke (WiSe 2021/2022)

Fabrice Beaumont
Matrikel-Nr: 2747609
Rheinische Friedrich-Wilhelms-Universität Bonn

November 4, 2021

Contents

Chapter 0

Motivations - Learning Tasks on Graphs

In this chapter we discuss different learning tasks on graph structured data.
A graph is a type of data structure which describes relationships (edges) between objects (vertices). For more formal definitions see definitions 0.2 and 0.1. Family trees, social network connections and molecules are examples for graphs.
Graphs may incorporate much information in form of attributes. Vertices for example may correspond to persons and may be attributed by name, age or gender. In this example edges may additionally contain information about relationship properties. Another example are molecular graphs, which have atom types as vertex labels and their edges indicate the type of covalent bond.
The three main learning paradigms of machine learning are:

- **Supervised Learning**: Find a data-consistent function that maps input to output values. In other words, given a set of training data $X = \{(x, f(x))\}$, find a hypothesis function $h \in \mathcal{H}$ such that $h \approx f$.
  (E.g. Linear Regression, Logistic Regression, Support Vector Machines, Nearest Neighbor, Naive Bayes Classifier, Decision Trees & Random Forests, (Deep) Neural Networks, ...)

- **Unsupervised Learning**: Find structures by identifying data correlations without explicit knowledge of memberships.
  (E.g. clustering, auto-encoders, (frequent) pattern mining, ...)

- **Reinforcement Learning**: Learn a function which maximizes the cumulative output rewards by choosing appropriate actions in an unknown environment.
  (Search space exploration to reason about the environment.)

Common supervised learning tasks on graphs are:

- Assign a color to unknown vertices
- Predict likely new edges
- Predict unknown graph class

Common unsupervised learning tasks on graphs are:

- Vertex clustering, Community detection
- Graph clustering
- Link prediction without supervision

The central problem of learning on graphs is how traditional machine learning approaches can be applied to graphs.
Linear Regression, Logistic Regression, Naive Bayes Classifiers, Decision trees and Random Forests expect *tabular data*, i.e. a set of example with a known, fixed set of features.
One may consider to use features of vertices to learn on them. But several non-trivial questions arise, for example:

- How to deal with connections between vertices?
- How to deal with vertices without features?
- What if the objects we want to learn from are graphs themselves? (Possibly with different numbers of vertices or edges.)

Note that the usage of a graph's adjacency matrix as vector representation is not desirable, since isomorphic graphs can be represented in different adjacency matrices.
Another approach to make traditional machine learning approaches applicable to graphs is to use **graph representations**. These vectorize data with respect to task specific properties. Generally, we seek approaches that incorporate aspects of the graph's structure.
It is highly non-trivial to determine which information crucial for a specific learning task shall be represented in which way in the graph representation.

Example - Vertex classification: Consider the graph depicted in Figure 1 and the task to reason about the color of unknown vertices by looking at training data.

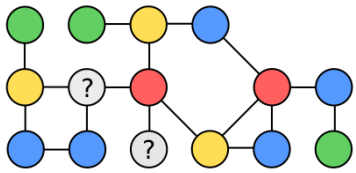

Figure 1 – Example of a graph with colors as vertex features.

A closer look reveals that, in this example, the color of a vertex corresponds to its degree. Thus $\deg(v)$ may be a simple representation of each vertex v.

Example - Graph classification: Consider the graph depicted in Figure 2 and the task to predict the class of an unclassified graph (e.g. the gray one in the figure).

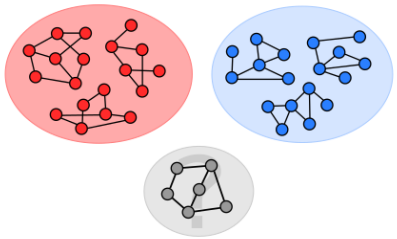

Figure 2 – Example of graph classification (clustering).

A common approach is to use sets of subgraphs to describe (similarity between) graphs. In this example it appears that graphs of the red class contain cycles of length four whereas graphs of the blue class contain triangles. Thus, a graph G may be represented by a vector $\phi(G)$ containing cycle counts of length three and four.

Example - Link prediction: Consider the graph depicted in Figure 3 and the task to predict the likelihood of a not jet existing edge (e.g. the gray one in the figure).

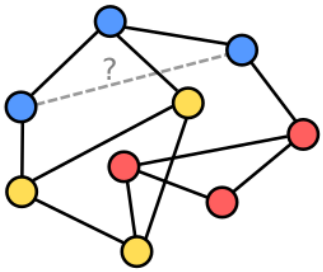

Figure 3 – Example of graph classification (clustering).

A common approach is to consider the shortest path distance and similarity in terms of neighborhoods or vertex attributes. In this example the vertices of same color are more likely to be connected by an edge. Thus one may represent edges using the adjacent vertex colors.
As the given examples indicate, it can be tedious to analyze graph data and choose a suitable graph representation. As it turns out, Machine Learning can be used for learning the graph representations themselves. In the following chapter we will cover two aspects of graph representation learning:

- **Learning with representations** of graphs and
- **learning representations** of graphs.

Chapter 1

Vertex classifications and local features

Definition 0.1: Directed graph

A **directed graph** (**digraph**) $G = (V, E)$ consists of a set of vertices $V(G)$ and a set of edges $E(G) \subseteq V \times V$.

Definition 0.2: Undirected Graph

A **undirected graph** $G = (V, E)$ consists of a set of vertices $V(G)$ and a set of edges $E(G) \subseteq 2^V$ such that

$$\forall e \in E(G): \qquad |e| = 2$$

As variable naming conventions, we say that graphs have $n = |V(G)|$ vertices and $m = |E(G)|$ edges.

Definition 0.3: Simple graphs

A simple graph is a graph, where edges connot appear multiple times.

(That is, in the respective definitions of directed and undirected graphs, the set of edges is not a multiset.)

Definition 0.4: Graph attributes and labels

Given an undirected or directed graph G, one can define additional data associated with the graph, vertices or edges. Such data is called **attributes** (if the value domain is continuous) or **labels** (if the value domain is discrete/categorical).
This association can be formalized using **feature spaces** $\mathcal{X}^G$, $\mathcal{X}^V$ and $\mathcal{X}^E$:

$$f_G(G) \to \mathcal{X}^G, \qquad f_V\big(V(G)\big) \to \mathcal{X}^V, \qquad f_E\big(E(G)\big) \to \mathcal{X}^E$$

(A feature space is some product space $\mathcal{X} = (X_1, X_2, \ldots, X_d)$ where the X_i are some appropriately chosen sets.)

Example - Graph labels Typical examples of graph vertex labels are atom names (chemical graphs).

Definition 0.5: Graph isomorphism

Let G and H be two graphs. G and H are **isomorphic**, if and only if there exists a bijective function between the vertices $\varphi : V(G) \to V(H)$ such that

- edges are preserved:
  $\{v, w\} \in E(G) \iff \{\varphi(v), \varphi(w)\} \in E(H)$
- vertex attributes are preserved:
  $\forall v \in V(G) : f_V(v) = f_V\big(\varphi(v)\big)$
- edge attributes are preserved:
  $\forall \{v, w\} \in E(G) : f_E(\{v, w\}) = f_E\big(\{\varphi(v), \varphi(w)\}\big)$

1.1 Property prediction for vertices

Learning functions (properties) on vertices is done under the assumption that there is a multiset of objects $X \subseteq \mathcal{X}$ and there exists an (partially) unknown function $y : X \to \mathcal{Y}$ and some pairs of objects are associated. This setting is formalized in definition 1.1.

Definition 1.1: Supervised vertex property prediction on fixed graphs

Input: a graph G with labeling functions f_V, f_E,
a hypothesis class $\mathcal{H} \subseteq \{h |\ h : V(G) \to \mathcal{Y}\}$,
properties $y(v)$ for a set of vertices $v \in V' \subseteq V(G)$ and
a loss function $L_{V,y} : \mathcal{H} \to \mathbb{R}$.
Output: $\operatorname{argmin}_{h \in \mathcal{H}} L_{V,y}(h)$.

Note that one can extend hypothesis classes on graph vertices to their vertex representation. More explicitly, let $\mathcal{X}$ be a feature space and $r : V(G) \to \mathcal{X}$ some vertex representation on the graph G. Let $\mathcal{H} \subseteq \{h |\ h : \mathcal{X} \to \mathcal{Y}\}$ be a hypothesis class. Then the *extended hypothesis class*

$$\mathcal{H}' := \{h \circ r |\ h \in \mathcal{H}\} \tag{Eq. 1}$$

is a hypothesis class on $V(G)$, too.
On top of that one can generalize the vertex representation r to a set of several graph representation $\mathcal{R} \subseteq \{r |\ r : V(G) \to \mathcal{X}\}$ and yield the further extended hypothesis class $\mathcal{H}' := \{h \circ r |\ r \in \mathcal{R},\ h \in \mathcal{H}\}$ on $V(G)$.

1.1.1 Loss Functions - MERGE & MOVE WITH THE STATISTIC SCRIPT

To measure the quality of hypothesis, with respect to vertex property predictions, one can use a variety of loss functions. Most of these can be generalized to other applications of hypothesis evaluation.

Definition 1.2: Classification Loss

The **classification loss** of a hypothesis on vertices to predict their properties is defined as the probability of false predictions:

$$L_{V,y}(h) := \mathbb{P}\big[y(v) \neq h(v)\big] = 1 - \frac{1}{|V(G)|} \sum_{v \in V(G)} \delta\big(y(v), h(v)\big) \tag{Eq. 2}$$

where

$$\delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

Definition 1.3: Regression

The **mean squared error** can be used as a loss function of a hypothesis on vertex property prediction. Using it to improve the quality of the found hypothesis is called **regression**:

$$L_{V,y}(h) := \mathbb{E}\Big[\big(y(x) - h(x)\big)^2\Big] = \frac{1}{|V(G)|} \sum_{v \in V(G)} \big(y(v), h(v)\big)^2 \qquad \text{(Eq. 3)}$$

When evaluating a (machine learning) model during training, we test a hypothesis on empirical data and optimize for the best solution. Both the optimization and the empirical data induce a bias on the estimate of the quality of the hypothesis. This bias can be reduced using **cross validation** (TODO - see Figure **??**.

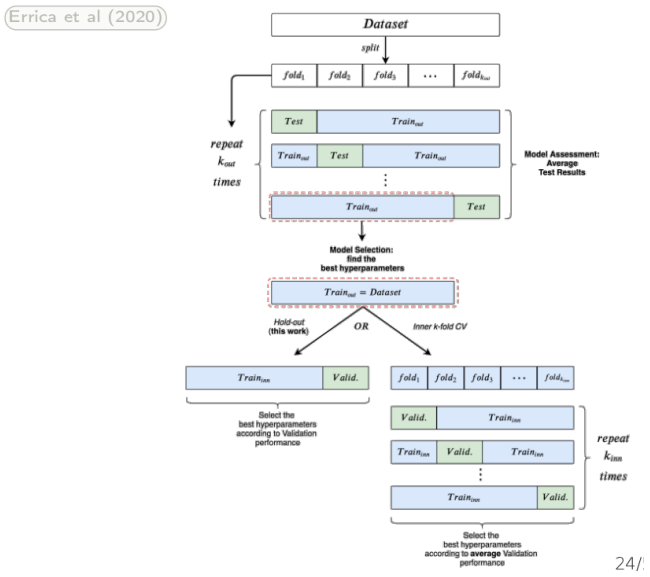

Figure 1.1 – TODO: replace image

Cross validation can be used to compare different hypothesis. If there is a statistically significant difference between their performances, one could be better than the other.

1.1.2 Baselines for graph learning

Interestingly, some baseline methods, which do not include specific graph structures may perform well, even better than GNNs on some graph datasets. In other words, improvements that do not clearly outperform structure-agnostic competitors should be viewed with moderation.
Note that whenever information about individual vertices is available, one should consider using it extensively. In many cases, it is more valuable for the task at hand, than the actual graph structure that combines these.
To conclude, to evaluate a graph representation one should measure its performance against a model which ignores the graph structure. Thus when dealing with an attributed graph G it can be beneficial to start with its vertex attributes $f_V : V(G) \to \mathcal{X}$ and first consider a hypothesis class over $\mathcal{X}$, only. Only after that, one may think about including graph representations into the hypothesis class.

1.1.3 Local Feature Counts

In this subsection we will define various simple vertex representations $f : V(G) \to \mathbb{R}$ for fixed graphs. Let $\mathcal{R}'$ be the set of all one-dimensional vertex representations. These representations can be grouped in the following way:

$$\mathcal{R} := \{\times_R : V(G) \to \mathbb{R}^{|R|} |\ R \subseteq \mathcal{R}'\} \tag{Eq. 4}$$

where $\times_R(v) = \big(r_1(v), r_2(v), \ldots, r_{|R|}(v)\big)$ for a vertex v and representations $r_i \in R$.

Definition 1.4: Vertex neighborhood

Let G be a undirected graph and $v \in V(G)$. The **neighborhood** of v is the set

$$N(v) := \{w|\ \{v, w\} \in E(G)\}$$

If G is directed, we define respectively the **outgoind neighborhood** as

$$N_{\text{out}}(v) := \{w|\ (v, w) \in E(G)\}$$

and the **incoming neighborhood** as

$$N_{\text{in}}(v) := \{w |\ (w, v) \in E(G)\}$$

THEOREM 1.1 - Full Neighborhood Representation:

By fixing an order of the vertices $V = \{v_1, \ldots, v_n\}$ and defining

$$A(v_j)[i] := \begin{cases} 1 & \text{if } v_i \in N(v_k) \\ 0 & \text{if } v_i \notin N(v_k) \end{cases} \qquad \text{(Eq. 5)}$$

one can represent a vertex by its neighborhood. This is called **Full Neighborhood Representation**.

The drawbacks of a Full Neighborhood Representation include:

- The space requirements grow quadratically with n.
- Learning algorithms that use this representation will likely overfit.
- It requires additional effort to check, whether a given vertex is already included in the representation.
- It requires additional effort to find neighbors of neighbors.

The first two drawbacks can be reduced, by so called Compressed Neighborhood Representations.

Definition 1.5: Vertex degree

Let G be a undirected graph and $v \in V(G)$ be a vertex. The **degree** is the number of edges incident to v, i.e.

$$\delta(v) := |N(v)|$$

if G is directed, we define respectively the **outdegree neighborhood** as the number of (incident) edges starting in v, i.e.

$$\delta_{\text{out}}(v) := |N_{\text{out}}(v)|$$

and the **indegree neighborhood** as the number of (incident) edges ending

in v, i.e.

$$\delta_{\text{out}}(v) := |N_{\text{out}}(v)|$$

The neighborhood $N(v)$ of a vertex v induces a subgraph $G[N(v)]$ with the vertices in $N(v)$ and all edges between these vertices:

$$G[N(v)] = \Big(N(v), \big\{\{w, w'\} \big| \, w, w' \in N(v)\big\}\Big)$$

The generalized definition is given in definition 1.7. This subgraph $G[N(v)]$ can have at most

$$\frac{\delta(v)\big(\delta(v)-1\big)}{2}$$

edges, since every of the $\delta(v)$ neighbors of v can have at most one edge, to all other $\delta(v)$ neighbors in $G[N(v)]$. Notice, not to count every edge twice in this argumentation. If there are exactly this many edges (without loops), the subgraph is *fully connected*.

Definition 1.6: Neighborhood density

The **density** of a vertex's neighborhood v is defined as the ratio of all existing edges and all possible edges (in a fully connected neighborhood):

$$c(v) := \frac{2\big|E\big(G[N(v)]\big)\big|}{\delta(v)\big(\delta(v)-1\big)}$$

Definition 1.7: Induced subgraph

Let G be a graph and $X \subseteq V(G)$ a set of its vertices. The **subgraph** of G **induced by** X is the graph H with

- $V(H) = X$ and
- $E(H) = \big\{\{v, w\} \big| \, \{v, w\} \in E(G) \ \wedge \ v, w \in X\big\}$

A **triangle graph** is a simple graph on three vertices with three edges.

Definition 1.8: Triangle subgraphs

Let $\mathcal{D}(v)$ be the set of all triangle subgraphs of a graph G that contain the vertex v.

Let $\Delta(v) := |\mathcal{D}(v)|$ be its carnality.

THEOREM 1.2 - Upper bound on the adjacent triangle count:

Let G be a graph and $v \in V(G)$. Then

$$\Delta(v) \leq \left|E(G[N(v)])\right|$$

Proof: Let $D \in \mathcal{D}(v)$ be an arbitrary triangle adjacent to $v \in V(G)$. Then $V(D) \subseteq N(v) \subseteq G[N(v)]$ since in a triangle all vertices are adjacent and thus all vertices in the triangle are adjacent to v (i.e. contained in its neighborhood). Notice, that there are two kind of edges in $G[N(v)]$, the ones incident to v ($v \in e$, there are $\delta(v)$ many of them) and these which are not ($v \notin e$). The ones incident to v can be part of at most two triangles in $\Delta(v)$, while the edges that are not incident to v can be part in exactly one triangle in $\Delta(v)$. That is because the two vertices that are adjacent to such an edge, together with v, completely define a triangle (if all edges are present). On the other hand, every triangle in $D(v)$ must contain exactly one edge, that is not incident to v (and two which are incident to v).
Therefore the number of triangles $\Delta(v)$ is equal to the number of edges in $G[N(v)]$ that are not incident to v, which is a fraction of all edges in $G[N(v)]$:

$$\begin{aligned}\Delta(v) &= \left|\{e \in E(G[N(v)])|\ v \notin e\}\right| \\ &\leq \left|\{e \in E(G[N(v)])|\ v \notin e\}\right| + \delta(v) \\ &= \left|E(G[N(v)])\right|\end{aligned}$$

(Note that in the second equation, the equality is only achieved, if v has no neighbors.) □

Definition 1.9: Induced Subgraph Isomorphism Problem

Input: two graphs G, H.
Output: the decision if H is isomorphic to an induced subgraph of G.

THEOREM 1.3 - Difficulty of the Ind. Subgraph Iso. Pb.:

The induced subgraph isomorphism problem is **NP-complete**.

Proof: Let K_n be a complete graph on n vertices. Then solving the induced subgraph isomorphism problem for K_n and G solves the k-Clique Problem for G, which is NP-complete itself. □

Notice, that if one could count the induced subgraphs for two graphs efficiently, one could decide the induced subgraph isomorphism problem. Hence *counting induced subgraphs is NP-hard*.

But if one keeps the size of the induced subgraphs fix, the problem becomes tractable.

1.1.4 Graphlet Counting

Definition 1.10: Graphlet

Small connected induced subgraphs are called **graphlets**.

We would like to count, for each vertex v, the number of graphlets of a certain type, that contain v. Lets call these **v-incident graphlets**.

Definition 1.11: Incident Induced Graphlets Count Problem

Input: a graph G and
a graphlet H of size k.
Output: the number of v-incident induced subgraphs isomorphic to H for all $v \in V(G)$:
$r_H(v) := \left|\{X \subseteq V |\ v \in X \ \wedge\ G[X] \overset{\text{iso.}}{\equiv} H\}\right|$

Algorithm 1 Brute Force - Counting Incident Graphlets

Input: a graph G and
a graphlet H of size k.
Output: the number of v-incident induced subgraphs isomorphic to H for all $v \in V(G)$.

1: $\mathrm{gc}(v) = 0$ for all $v \in V(G)$
2: **for all** $X \subseteq V(G)$ with $|X| = k$ **do**
3: **if** $G[X]$ is isomorphic to H **then**
4: Increment $\mathrm{gc}(x)$ for all $x \in X$

Runntime: There are $\binom{n}{k}$ choices for X, and there need $\gamma(k) \in \mathcal{O}(k!\, k^2)$ graphs to be checked for isomorphism. Thus the total runtime for a constant k is

$$\mathcal{O}\big(n^k\, \gamma(k)\big) = \mathcal{O}(n^k)$$

Since a lot of graph structured data can be considered Big Data (w.r.t. size; not speed), cubic runtime in the number of vertices is not acceptable.

THEOREM 1.4 - Runtime bound on the Incident Induced Graphlets Count Problem:
Let G be a bounded degree graph and let d denote the maximum degree in G. Then all v-incident graphlets of G with size $k \in \{3, 4, 5\}$ can be enumerated in $\mathcal{O}(nd^{k-1})$ time.

Notice that the maximum degree d of a graph is typically much smaller than the number of its vertices n.

Proof: Graphlets of size k can be divided into two classes:

- Graphlets that contain a simple path of length $k - 1$ and
- Graphlets that to not contain such a path.

□

Chapter 2

Announced Topics

2.1 Link prediction

2.2 Graph classification, regression and clustering in transactional graph databases

Chapter 3

Feature Extraction and Graph Mining

In this chapter we focus on extracting graph representations following a fixed process. Therefore we discuss several pattern (or feature) classes and focus on the following questions:

- What kind of features are there (e.g. local and global)?
- How viable are these features?
- How expressive is their extraction?
- How powerful are graph representations in terms of distinguishing inputs?
- How do we evaluate their quality?

Chapter 4

Learning on Graphs and Graph Kernels

In this chapter we discuss learning with (implicit) graph representations. Therefore we introduce machine learning models specific to the task of classifying graphs.

4.1 Support Vector Machine (SVM)

4.2 Expressive power of machine learning models

4.3 Kernel of special interest: Weisfeiler-Lehman kernel

Chapter 5

Graph Neural Networks (GNNs)

In this chapter we will introduce the learning of representations of graphs. We discuss how neural networks can be used to learn graph representations and to subsequently perform machine learning tasks.

5.1 Relationship to Weisfeiler-Lehman method

5.2 Structure of GNNs

5.3 Expressive power of GNNs

Chapter 6

Exercises

6.1 Sheet 0 - Python

6.2 Sheet 1

6.2.1 Assignment 1a - Bias of an estimator

...

Solution:

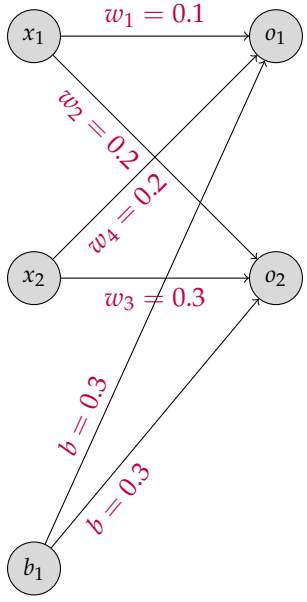

Figure 6.1 – Multi-Layer Perceptron

n	x_1	x_2	p_1	p_2
1	0.1	0.4	0.1	0.9
2	0.8	0.2	0.95	0.05
3	0.6	0.5	0.4	0.6
4	0.3	0.9	0.75	0.25
5	0.3	0.5	0.9	0.1

Table 6.1 – Training batch

Algorithm 2 Generative Adversarial Networks (GAN)

Input: a discriminator function D,
a generator function G,
noise samples Z,
true samples X (with an underlying data generation distribution $\mathbb{P}_{\text{data}}[X]$),
desired size of mini-batches m,
a number k of improvement iterations for the discriminator,
stopping criterion $T_{\text{condition}}$

Output: a generator G that is able to fool the trained discriminator D

1: **while** not $T_{\text{condition}}(f, \theta)$ **do**
2: **for** k steps **do**
3: Sample a minibatch of noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$
4: Sample a minibatch of true samples $\{x^{(1)}, \ldots, x^{(m)}\}$
5: Perform stochastic gradient ascend for the discriminator:

$$\nabla_{\theta^{(d)}} \frac{1}{m} \sum_{i=1}^{m} \Big(\log\big(d(x^{(i)}; \theta^{(d)})\big) + \log\big(1 - d\big(g(z^{(i)}; \theta^{(g)}), \theta^{(d)}\big)\big)\Big)$$

6: Sample a minibatch of noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$
7: Perform stochastic gradient descending for the generator:

$$\nabla_{\theta^{(g)}} \frac{1}{m} \sum_{i=1}^{m} \Big(\log\big(1 - d\big(g(z^{(i)}; \theta^{(g)}), \theta^{(d)}\big)\big)\Big)$$

Bibliography

[1] „EXAMPLE: Finite Elemente - Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie “von Dietrich Braess, 5. Auflage, Springer, 2013