Learning to Aggregate on Structured Data

Clemens Damke

February 16, 2020

PADERBORN UNIVERSITY
The University for the Information Society
Department of Electrical Engineering,
Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn

INTELLIGENT SYSTEMS
Intelligent Systems Group (ISG)

Master Thesis

Learning to Aggregate on Structured Data

Clemens Damke

1. Reviewer Prof. Dr. Eyke Hüllermeier
Intelligent Systems and Machine Learning Group (ISG)
Paderborn University

2. Reviewer Prof. Dr. Axel-Cyrille Ngonga Ngomo
Data Science Group (DICE)
Paderborn University

February 16, 2020

Clemens Damke
Learning to Aggregate on Structured Data
Master Thesis, February 16, 2020
Reviewers: Prof. Dr. Eyke Hüllermeier and Prof. Dr. Axel-Cyrille Ngonga Ngomo
Supervisor: Vitalik Melnikov

Paderborn University
Intelligent Systems and Machine Learning Group (ISG)
Heinz Nixdorf Institute
Department of Electrical Engineering, Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn

Abstract

Contents

1 Introduction

1.1 Motivation

The field of *machine learning* (ML) on graph-structured data has applications in many domains due to the general expressive power of graphs. The three most common types of graph ML problems are

1. **Link prediction:** A graph with an incomplete edge set is given and the missing edges have to be predicted. The suggestion of potential friends in a social network is a typical example for this.

2. **Vertex classification & regression:** Here a class or a score has to be predicted for each vertex of a graph. In social graphs this corresponds to the prediction of properties of individuals, e.g. personal preferences or gender. Another example is the prediction of the amount of traffic at the intersections of a street network.

3. **Graph classification & regression:** In this final problem type a single global class or continuous value has to be predicted for an input graph. The canonical example for this is the prediction of properties of molecule graphs, e.g. the toxicity or solubility of a chemical.

In this thesis we will focus on the last problem type, *graph classification and regression* (GC/GR). An ML method for this problem has to accept graphs of varying size and should be permutation invariant wrt. the order in which graph vertices are provided. Those requirements are not met by the commonly used learners that only accept fixed-size feature vectors as their input, e.g. *logistic regression model*s (LRMs), *support vector machine*s (SVMs) or multilayer perceptrons.

A GC/GR method has to account for two central aspects of the problem: 1. Local structural analysis and 2. global aggregation. The first aspect is about the extraction of relevant features of substructures of the input graph. The latter is about the way in which the local features are combined into a final class or regression value. The existing GC/GR methods are mostly motivated by local structural graph analysis. The aspect of global aggregation on the other hand is less emphasized by those methods.

There is however a separate branch of research that specifically looks at the problem of learning aggregation functions, called *learning to aggregate* (LTA). Current LTA

approaches explicitly learn an aggregation functions for sets which can be interpreted as graphs without edges. The motivation for this thesis is to generalize LTA from sets to arbitrary graphs. The overall goal is to combine the aggregation learning perspective with existing GC/GR methods.

1.2 Goals

To extend LTA to graphs, three goals have to be achieved:

1. **Formalization of LTA:** Before LTA can be extended, its essential characteristics have to be defined. Those characteristics should provide the terminology to formally capture the differences and similarities between LTA and existing GC/GR methods.

2. **Give an LTA interpretation of GC/GR methods:** Using the LTA formalization, representative GC/GR approaches should be restated as LTA instances. Currently there is no comprehensive formulation of the relation between both fields of research; this is addressed by the the second goal.

3. **Define an LTA method for graphs:** Using the LTA perspective on GC/GR, hidden assumptions of the existing approaches should become clear and in which way they share the assumptions of LTA. The last goal is to use those insights to formulate an LTA-GC/GR method that combines ideas from the existing approaches with the LTA assumptions.

1.3 Structure

Chapter 2: Related Work

Chapter 3: Learning to Aggregate on Graphs

Chapter 4: Evaluation

Chapter 5: Conclusion

2 Related Work

Before combining LTA and GC/GR as described in section 1.2, we first give an overview of the state-of-the-art in both fields of research. This is done in three steps:

1. We begin with an overview of the existing LTA methods for unstructured inputs.

2. Then we look at the domain of structured inputs. To solve the GC/GR problem, relevant graphs characteristics have to be defined in order to determine the similarity and dissimilarity of graphs. We will look at two common approaches for graph characterization: The Weisfeiler-Lehman algorithm and the notion of graph spectra.

3. Using the described graph characterization approaches, a brief overview of current GC/GR methods will then be given.

2.1 Learning to Aggregate

The class of LTA problems was first described by Melnikov and Hüllermeier [MH16]. There an input instance is understood as a composition $\boldsymbol{c} = \{\!\{c_1, \ldots, c_n\}\!\}$ of so-called constituents, i.e. as a variable-size multiset (denoted as $\{\!\{ \cdot \}\!\}$). The assumption in LTA problems is that for all constituents $c_i \in \boldsymbol{c}$ a local score $y_i \in \mathcal{Y}$ is either given or computable. The set of those local scores should be indicative of the overall score $y \in \mathcal{Y}$ of the composition $\boldsymbol{c}$. LTA problems typically require two subproblems to be solved:

1. **Aggregation:** A variadic aggregation function $\mathcal{A} : \mathcal{Y}^* \to \mathcal{Y}$ that estimates composite scores has to be learned, i.e. $y_i \approx \hat{y} = \mathcal{A}(y_1, \ldots, y_n)$. Typically the aggregation function $\mathcal{A}$ should be associative and commutative to fit with the multiset-structure of compositions.

2. **Disaggregation:** In case the constituent scores y_i are not given, they have to be derived from a constituent representation, e.g. a vector $x_i \in \mathcal{X}$. To learn this derivation function $f : \mathcal{X} \to \mathcal{Y}$, only the constituent vectors $\{\!\{x_i\}\!\}_{i=1}^n$ and the composite score y is given. Thus the constituent scores y_i need to be *disaggregated* from y in order to learn f.

Overall LTA can be understood as the joint problem of learning the aggregation function $\mathcal{A}$ and the local score derivation function f. Two main approaches to

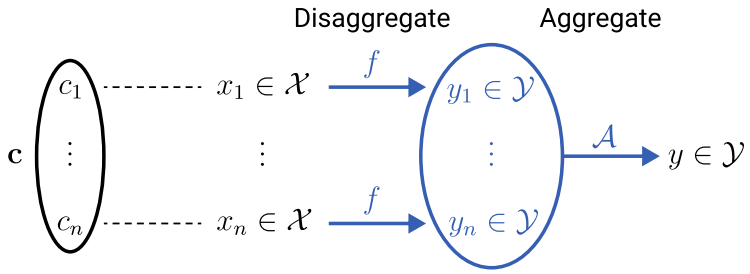


Figure 2.1. Overview of the structure of LTA for multiset compositions.

represent the aggregation function in LTA problems have been explored.

2.1.1 Uninorm-Aggregation

The first approach uses *uninorms* [MH16] to do so. There the basic idea is to express composite scores as fuzzy truth assignments $y \in [0,1]$. Such a composite assignment y is modeled as the result of a parameterized logical expression of constituent assignments $y_i \in [0,1]$. As the logical expression that thus effectively aggregates the constituents, a uninorm U_λ is used. Depending on the parameter $\lambda \in [0,1]$, U_λ combines a t-norm T and a t-conorm S which are continuous generalizations of logical conjunction and disjunction respectively. One popular choice of norms are the so-called Łukasiewicz norms:

$$\text{t-norm } T(a,b) := \max\{0, a+b-1\}, \quad \text{t-conorm } S(a,b) := \min\{a+b, 1\},$$

$$\text{uninorm } U_\lambda(a,b) := \begin{cases} \lambda T\left(\frac{a}{\lambda}, \frac{b}{\lambda}\right) & \text{if } a,b \in [0,\lambda] \\ \lambda + (1-\lambda) S\left(\frac{a-\lambda}{1-\lambda}, \frac{b-\lambda}{1-\lambda}\right) & \text{if } a,b \in [\lambda,1] \\ \lambda \min\{a,b\} & \text{else} \end{cases} \tag{2.1}$$

At the extreme points $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$, T and S coincide with the Boolean operators $\wedge$ and $\vee$; the values at all other points are interpolated as shown in fig. 2.2. The uninorm U_λ uses the conjunctive t-norm T for values below the threshold λ and the disjunctive t-conorm S for values above the threshold. U_λ therefore smoothly interpolates between a conjunctive and disjunctive operator with the extreme points $U_1 = T$ and $U_0 = S$.

Since t-norms and t-conorms are commutative and associative they can also be applied to non-empty sets of arbitrary size, i.e. $T(\{\!\{y_1, \ldots, y_n\}\!\}) = T(y_1, T(\{\!\{y_2, \ldots, y_n\}\!\}))$ with fixpoint $T(\{\!\{y\}\!\}) = y$. Using this extension, a uninorm U_λ can be applied to sets which turns it into a parameterized aggregation function $\mathcal{A}_\lambda : [0,1]^* \to [0,1]$. In this simple model the LTA aggregation problem boils down to the optimization of λ. The LTA disaggregation problem is solved by jointly optimizing a *logistic regression model* (LRM), i.e. the constituent scores $\{\!\{y_i \in [0,1]\}\!\}_{c_i \in \mathbf{c}}$ are described by $y_i = \left(1 + \exp(-\theta^\top x_i)\right)^{-1}$. Overall an LTA model is therefore described by the

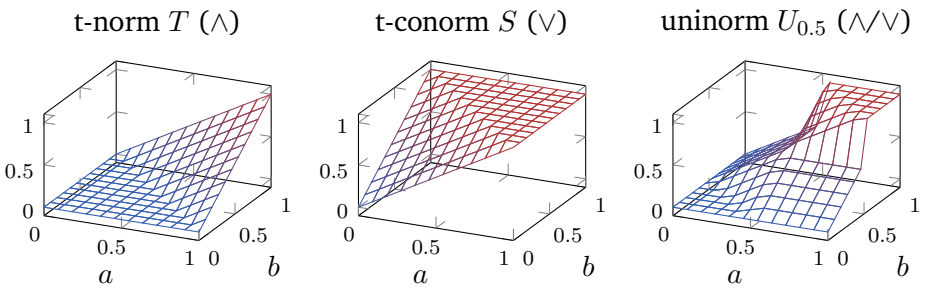


Figure 2.2. The Łukasiewicz norms and the corresponding uninorm for $\lambda = 0.5$.

uninorm parameter λ and the regression coefficients θ.

2.1.2 OWA-Aggregation

Recently Melnikov and Hüllermeier [MH19] have looked at an alternative class of aggregation functions. Instead of using fuzzy logic to describe score aggregation, *ordered weighted average* (OWA) operators were used. OWA aggregators work by sorting the input scores and then weighting them based on their sort position, i.e.

$$\mathcal{A}_\lambda(y_1, \ldots, y_n) := \sum_{i=1}^{n} \lambda_i y_{\pi(i)}, \tag{2.2}$$

where $\lambda \in \mathbb{R}^n$ is a weight vector with $\|\lambda\|_1 = 1$ and π is a sorting permutation of the input scores with $y_i < y_j \Rightarrow \pi(i) < \pi(j)$. Depending on the choice of the vector λ, the OWA function $\mathcal{A}_\lambda$ can express common aggregation functions like $\min$ (if $\lambda = (1, 0, \ldots, 0)$), $\max$ (if $\lambda = (0, \ldots, 0, 1)$) or the arithmetic mean (if $\lambda = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$).

To deal with varying composition sizes n, the weights $\lambda_1, \ldots, \lambda_n$ can however not be statically assigned. Instead they are interpolated using a so-called *basic unit interval monotone* (BUM) function $q : [0, 1] \to [0, 1]$. It takes constituent positions that are normalized to the unit interval, i.e. $\frac{i}{n} \in [0, 1]$. The BUM function q is then used to interpolate a weight for any normalized sort position via $\lambda_i := q\left(\frac{i}{n}\right) - q\left(\frac{i-1}{n}\right)$. Because q is monotone with $q(0) = 0$ and $q(1) = 1$, it always holds that $\|\lambda\|_1 = q(1) - q(0) = 1$. Using this model, the aggregation problem boils down to optimizing the shape of q.

In the OWA approach the BUM function q is modeled as a piecewise linear spline. This spline is described by $m + 1$ points $\left\{\left(\frac{j}{m}, a_j\right)\right\}_{j=0}^{m}$, the so-called knots of the spline. The curve of q is obtained by linearly interpolating between neighboring knots as shown in fig. 2.3. If $0 = a_0 \leq a_1 \leq \cdots \leq a_m = 1$, q is a BUM function. The LTA aggregation problem is therefore solved by optimizing $a \in \mathbb{R}^{m+1}$ under this constraint. The disaggregation problem is tackled by adding the scores $y_1, \ldots, y_M \in \mathbb{R}$ to the learnable parameters of the model where M is assumed to be the finite number of

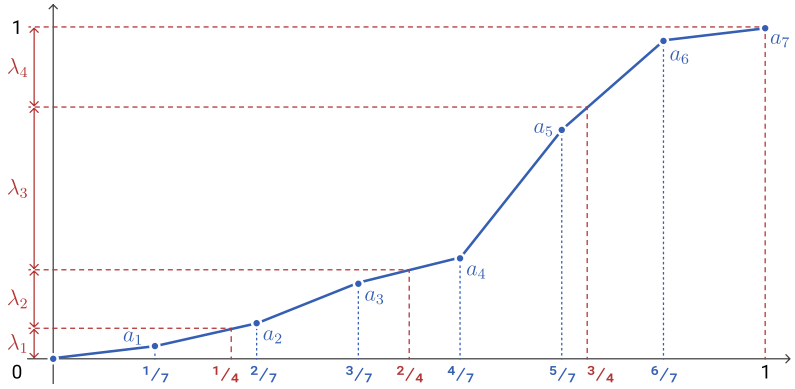


Figure 2.3. Illustration of how a describes q and its relation to λ ($n = 4$, $m = 7$).

constituents. Currently the OWA approach requires all possible constituents to be part of the training dataset since it does not consider constituent features $x_i \in \mathcal{X}$ to predict the scores of previously unseen constituents.

2.2 Graph Characterization

To classify or score a graph, it first needs to be characterized by a set of relevant properties. Two commonly used graph characterization approaches are the so called Weisfeiler-Lehman coloring and the notion of the graph spectrum. Both will now be briefly introduced since they form the foundation of most current GC/GR methods.

2.2.1 Weisfeiler-Lehman Graph Colorings

The *Weisfeiler-Lehman* (WL) algorithm [WL68] characterizes a graph $G = (\mathcal{V}, \mathcal{E})$ by assigning discrete labels $c \in \mathcal{C}$, called *colors*, to vertex k-tuples $(v_1, \ldots, v_k) \in \mathcal{V}^k$, where $k \in \mathbb{N}$ is the freely choosable *WL-dimension*. A mapping $\chi_{G,k} : \mathcal{V}^k \to \mathcal{C}$ is called a *k-coloring* of $G = (\mathcal{V}, \mathcal{E})$. We say χ' *refines* χ ($\chi' \trianglerighteq \chi$) iff. $\forall a, b \in \mathcal{V}^k : \chi(a) \neq \chi(b) \to \chi'(a) \neq \chi'(b)$, i.e. χ' distinguishes at least those tuples that are distinguished by χ.

The k-dimensional WL algorithm (k-WL) works by iteratively refining k-colorings $\chi^0_{G,k} \trianglelefteq \chi^1_{G,k} \trianglelefteq \ldots$ of a given graph G until the convergence criterion $\chi^i_{G,k} \equiv \chi^{i+1}_{G,k}$ is satisfied; here the equivalence operator $\equiv$ denotes that $\chi^i_{G,k}$ and $\chi^{i+1}_{G,k}$ must be identical up to color/label substitutions. The final WL coloring $\bar{\chi}_{G,k}$ that is obtained after convergence can be used to quickly check whether two graphs are not isomorphic. If for two graphs G and H there exists a color $c \in \mathcal{C}$ s.t. $\left|\left\{a \in \mathcal{V}^k_G \mid \bar{\chi}_{G,k}(a) = c\right\}\right| \neq \left|\left\{b \in \mathcal{V}^k_H \mid \bar{\chi}_{H,k}(b) = c\right\}\right|$, the graphs are k-WL *distinguishable* ($G \not\simeq_k H$) and therefore non-isomorphic ($G \not\simeq H$). The opposite does however not necessarily hold; two k-WL indistinguishable graphs are not always isomorphic, i.e. $G \simeq_k H \not\Longrightarrow G \simeq H$. To understand why this is the case and which types of graphs are k-WL distinguishable,

we will now look at the details of WL color refinement step. First the color refinement algorithm for the most simple case of $k = 1$ is described. Then the definitions and intuitions from the 1-dimensional case are extended to its higher-dimensional generalization. Lastly we will discuss the discriminative power of the WL algorithm and its relation to the WL-dimension k.

The 1-dimensional WL algorithm

In the 1-dimensional WL algorithm, a color is assigned to each vertex of a graph. If the vertices $\mathcal{V}_G$ of the input graph G are already labeled via some labeling $l_G : \mathcal{V}_G \to L$, those labels can be used as the initial graph coloring $\chi^0_{G,1} = l_G$. For unlabeled graphs a constant coloring is used instead, e.g. $\forall v \in \mathcal{V}_G : \chi^0_{G,1}(v) = \texttt{A}$ for some initial color $\texttt{A} \in \mathcal{C}$. In each iteration of the 1-WL color refinement algorithm, the following neighborhood aggregation scheme is used to compute a new color for all vertices:

$$\chi^{i+1}_{G,1}(v) := h\left(\chi^i_{G,1}(v), \{\!\{\chi^i_{G,1}(u) \mid u \in \Gamma_G(v)\}\!\}\right), \tag{2.3}$$

with $\Gamma_G(v)$ denoting the set of adjecent vertices of $v \in \mathcal{V}_G$ and $h : \mathcal{C}^* \to \mathcal{C}$ denoting a perfect hash function that assigns a unique color to each finite combination of colors. In practice h is usually defined lazily by using $\mathcal{C} = \mathbb{N}$ and non-deterministically enumerating color combinations in the order in which they are hashed s.t. a new color is introduced every time a previously unseen color combination is queried.

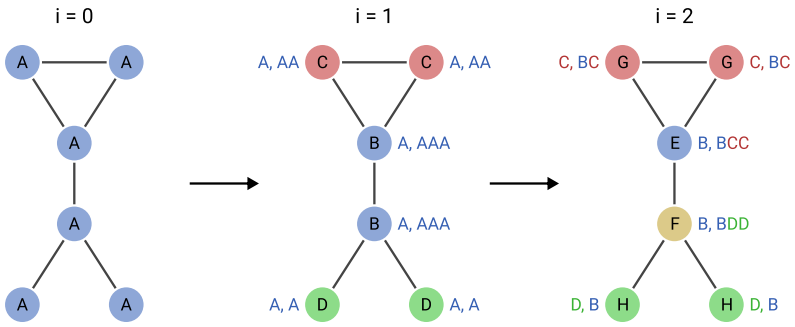


Figure 2.4. Example 1-WL color refinement steps. After two iterations the coloring stabilizes. Each vertex is labeled with its current color and has the aggregated neighbors, as defined in eq. (2.3), written next to it.

The k-dimensional WL algorithm

As we just saw, the 1-WL algorithm iteratively refines colorings of single vertices. While the obtained colorings differ for most non-isomorphic graphs $G \not\simeq H$, 1-WL does not generally solve the graph isomorphism problem as illustrated in fig. 2.5. By extending WL to higher dimensions such 1-WL indistinguishable cases can however be

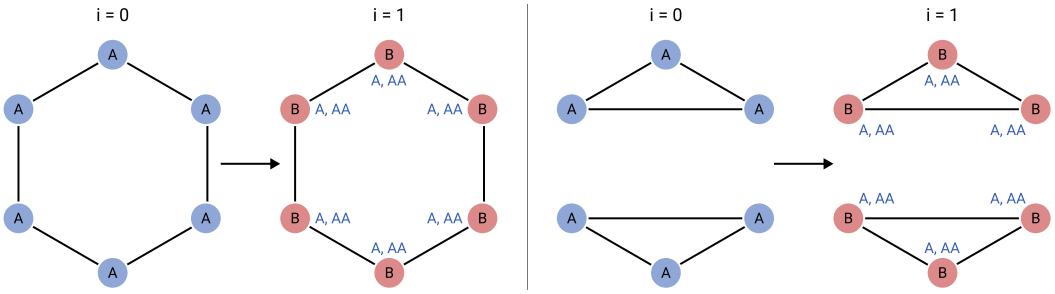
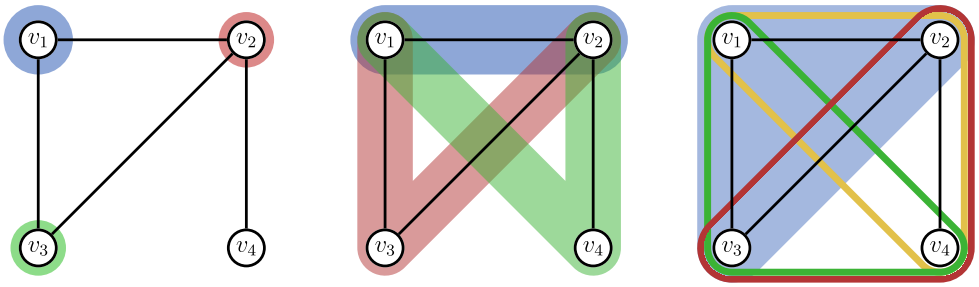


Figure 2.5. Two simple non-isomorphic graphs that are indistinguishable by 1-WL.

handled. Analogous to the 1-dimensional definition from eq. (2.3), the k-dimensional color refinement step is defined by

$$\chi^{i+1}_{G,k}(v) := h\left(\chi^{i}_{G,k}(v), \{\!\{(\chi^{i}_{G,k}(v[u/1]), \ldots, \chi^{i}_{G,k}(v[u/k])) \mid u \in \mathcal{V}_G\}\!\}\right) \tag{2.4}$$
$$\text{with } v = (v_1, \ldots, v_k) \in \mathcal{V}^k \quad \text{and} \quad v[u/j] := (v_1, \ldots, v_{j-1}, u, v_{j+1}, \ldots, v_k).$$

In 1-WL a vertex color is refined by combining the colors of neighboring vertices. In k-WL the color of a k-tuple $v \in \mathcal{V}^k$ is refined by combining the colors of its neighborhood which is defined as the set of all k-tuples in which at most one vertex differs from v. Note that each vertex k-tuple has one neighbor for each $u \in \mathcal{V}_G$, each of which is a k-tuple of vertex k-tuples. This more abstract notion of neighborhood is illustrated in fig. 2.6. For $k = 2$ this means that each potential edge $(v, w) \in \mathcal{V}_G^2$ has



(a) 1-WL v_1 neighbors **(b)** 2-WL (v_1, v_2) neighbors **(c)** 3-WL (v_1, v_2, v_3) neighbors

Figure 2.6. Tuple neighborhoods for different values of k. The vertices highlighted in blue form the root tuple whose neighbors are shown. Each neighbor is highlighted with a different color, except for 3-WL where the red, green and yellow triples actually form the single neighbor for $u = v_4$ (see eq. (2.4)).

all possible paths of length 2 from v to w as its neighbors (see fig. 2.6b). Also note that, even though k-WL refines k-tuple colors, lower-dimensional structures still get their own colors since a tuple does not have to consist of distinct vertices, i.e. in k-WL the color of a single vertex $v \in \mathcal{V}_G$ is described by $\bar{\chi}_{G,k}(s)$ for $s = (v, \ldots, v) \in \mathcal{V}_G^k$.

Let us now look at how the tuple colors are initialized. For this we use subgraph *isomorphism types* which determine the initial color $\chi^0_{G,k}(v)$ of each k-tuple v. For $k = 1$ the isomorphism type of a vertex v directly corresponds to its label $l_G(v)$. For

$k > 1$ the isomorphism types more generally correspond to the classes of isomorphic ordered subgraphs of G induced by the k-tuples $\mathcal{V}_G^k$. Formally this means that

$$\begin{aligned} \chi^0_{G,k}(v_1,\ldots,v_k) = \chi^0_{G,k}(w_1,\ldots,w_k) \iff & \forall i : l_G(v_i) = l_G(w_i) \\ & \wedge \forall i,j : v_i = v_j \leftrightarrow w_i = w_j \\ & \wedge \forall i,j : (v_i,v_j) \in \mathcal{E}_G \leftrightarrow (w_i,w_j) \in \mathcal{E}_G. \end{aligned} \tag{2.5}$$

If no explicit vertex labeling l_G is given, a single constant label for all vertices is assumed. Also note that there is a fundamental difference in how the adjacency information encoded in $\mathcal{E}_G$ is used in 1-WL vs. k-WL: In 1-WL a vertex coloring by itself cannot encode adjacency which is why this information is explicitly incorporated in each refinement step via Γ_G (see eq. (2.3)). In k-WL on the other hand each pair of vertices $(v,u) \in \mathcal{V}_G^2$ appears in at least one k-tuple (assuming $k \geq 2$) and therefore has at least one color which can implicitly encode the adjacency information. Edges and non-edges are colored differently in the initial coloring as defined in eq. (2.5); thus no explicit adjacency information is needed in the k-WL color refinement step defined in eq. (2.4).

Discriminative Power of WL

Now we will look at the types of graphs that can be distinguished by WL in relation to the WL-dimension k.

Theorem 2.1. $G \not\simeq_k H \implies G \not\simeq_{k+1} H$, i.e. the discriminative power of k-WL grows with k.

Proof Sketch. All k-tuples can be mapped to $(k+1)$-tuples, e.g. via $\varphi(v_1,\ldots,v_k) = (v_1,\ldots,v_k,v_k)$. □

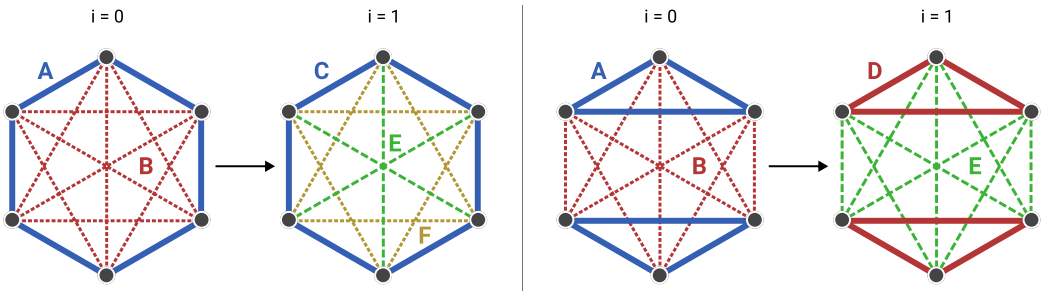


Figure 2.7. Two simple non-isomorphic graphs that are indistinguishable by 1-WL.

2.2.2 Spectral Graph Theory

2.3 Graph Classification and Regression

The existing approaches to tackle the GC/GR problem can be categorized into three main families: 1. Explicit graph embeddings, 2. graph kernels and 3. graph neural networks. We will now look at the characteristics of those families and give a brief overview of specific methods.

2.3.1 Explicit Graph Embeddings

The basic idea of explicit graph embedding approaches is to map a graph $G \in \mathbb{G}$ to some vector in a finite vector space $\mathcal{X} = \mathbb{R}^d$. A function $f : \mathbb{G} \to \mathcal{X}$ is called a *graph embedding function*. By embedding a graph into $\mathcal{X}$, any classification or regression algorithm that works with vectors can then be applied to solve the GC/GR problem.

The so-called vertex embedding problem is closely related to the graph embedding problem. As the name suggests, a *vertex embedding function* $f_G : V \to \mathcal{X}$ maps all vertices $v \in V$ of a graph G into $\mathcal{X}$. The embedding vector $f_G(v)$ ideally encodes relevant information about a vertex and its structural position in G. It can be used to solve the vertex classification and regression problem via arbitrary ML methods for vectors. We will now look at two main families of explicit graph and vertex embedding approaches.

Fingerprint Embeddings

The first works on graph embeddings were motivated by the study of chemical structures [AB73][WW86]. There a molecule can be interpreted as a labeled graph for which the GC/GR problem corresponds to the prediction of some chemical property, e.g. toxicity or solubility. So-called *fingerprint embeddings* try to match a fixed set of subgraphs $S_1, \ldots, S_d$ to the input graph. The embedding of a graph G is a binary vector $f(G) = x \in \{0,1\}^d$ with $x_i = \mathbb{1}[S_i \text{ is subgraph of } G]$, where $\mathbb{1}$ denotes the indicator function. This simple approach usually requires a careful choice of subgraphs but can still be competitive with the other more recent approaches we will look at in the following sections. Fingerprint embeddings are for example used in multiple state-of-the-art toxicity prediction tools like RASAR [Lue+18][Tox], ProTox [Drw+14][Ban+18b][Ban+18a] or the Toxicity Estimation Software Tools [Tes].

Skip-gram inspired Embeddings

Skip-gram embeddings were introduced by Mikolov et al. as part of the well-known `word2vec` [Mik+13] word embedding method from natural language processing.

While a fingerprint embedding explicitly assigns an interpretation to each embedding dimension (i.e. to each standard basis vector), a skip-gram embedding only optimizes the distance between embedding vectors based on the similarity of the embedded instances without providing an interpretation of the embedding dimensions.

word2vec Let us first look at the `word2vec` skip-gram method. It gets a sequence of words $(w_0, \ldots, w_n)$ as input and outputs embedding vectors $f(w_0), \ldots, f(w_n) \in \mathbb{R}^d$. To do this the context $C_k(w_i) = \{w_{i-k}, \ldots, w_{i+k}\}$ is computed for all words where w_i is the so-called *context root*. The word contexts are then used to optimize the following log-likelihood objective:

$$\max_{f, f_C} \sum_{i=1}^{n} \log P(C_k(w_i)|w_i) \;=\; \max_{f, f_C} \sum_{i=1}^{n} \sum_{w_j \in C_k(w_i)} \left[f(w_i)^\top f_C(w_j) - \log Z_{w_i} \right] \tag{2.6}$$

$$\text{with } P(C_k(w_i)|w_i) := \prod_{w_j \in C_k(w_i)} \overbrace{\frac{\exp\left(f(w_i)^\top f_C(w_j)\right)}{Z_{w_i}}}^{P(w_j|w_i)} \text{ and } Z_{w_i} := \sum_{j=1}^{n} \exp\left(f(w_i)^\top f_C(w_j)\right)$$

`word2vec` essentially uses an expectation maximization scheme to maximize the probabilities $P(w_j|w_i)$ of observing the context words $w_j \in C_k(w_i)$ of all words w_i. Those probabilities are described by the overlap of the embeddings $f(w_i)$ of words w_i and the embeddings $f_C(w_j)$ of their context words w_j. Intuitively this means that words with similar contexts will be mapped close to each other in the embedding space. Note that `word2vec` actually finds two embeddings $f(w)$ and $f_C(w)$ for each word of which only the first is returned. The two embeddings represent two different perspectives on words: f describes a word w_i as the root of a context $C_k(w_i)$, f_C on the other hand describes a word w_j as part of a context $C_k(w_i) \ni w_j$.

Vertex Embeddings Skip-gram embeddings can be naïvely extended to graphs by realizing that `word2vec` effectively already is a vertex embedding method for linear graphs in which $C_k(v)$ is simply the k-neighborhood of the vertex/word v. The problem with this naïve extension is that the sizes of k-neighborhoods in arbitrary graphs can be much larger and often tend to grow exponentially with k. To deal with this computational problem the so-called DeepWalk [Per+14] and `node2vec` [GL16] methods perform random walks of fixed length to effectively take samples from the neighborhood of vertices. Both methods only differ in the transition matrix that is used for the random walk. Another difference of DeepWalk and `node2vec` compared to `word2vec` is the so-called *feature space symmetry* which states that the context root interpretation (f) of a vertex should be symmetric to its context element interpretation (f_C), i.e. $f = f_C$. The combination of random walk context sampling and and the feature space symmetry assumption can be used to compute vertex embeddings even for very large graphs.

Graph Embeddings Skip-gram methods can not only be used for vertex embeddings but also to embed entire graphs. One way to do this is via the `graph2vec` [Nar+17] method. It is inspired by `doc2vec` [LM14] which in turn is based on `word2vec`. `graph2vec` gets a set of graphs $\mathbb{G} = \{G_1, \dots G_N\}$ as input and outputs graph embeddings $f(G_1), \dots, f(G_N)$. While in `word2vec` every word can be a context root as well as a context element, `graph2vec` uses the graphs $\mathbb{G}$ as context roots. The context $C_k(G_i)$ of a graph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ is defined as

$$C_k(G_i) := \bigcup_{v_j \in \mathcal{V}_i} \left\{ \text{WL}_l(G_i)_j \right\}_{l=0}^{k} \text{ with } \text{WL}_l(G) := l \text{ iter. WL-1 coloring of } G. \tag{2.7}$$

Intuitively this context can be understood as the set of WL-distinguishable subgraphs of G_i with diameter $\leq 2k$. Since WL is used to identify distinct subgraphs, `graph2vec` can only be applied to graphs with discrete vertex labels. Using the previous definitions, the context root embedding function has the signature $f : \mathbb{G} \to \mathbb{R}^d$ while the context element embedding function is of type $f_C : \bigcup_{G_i \in \mathbb{G}} C_k(G_i) \to \mathbb{R}^d$. To find those embeddings the `word2vec` objective from eq. (2.6) is reused. Analogous to `word2vec`, `graph2vec` therefore embeds graphs that share subgraphs close to each other, whereas graphs that do not share substructures tend to be embedded further away from each other.

2.3.2 Graph Kernels

Instead of explicitly mapping a graph or vertices into a vector space, one can also do so implicitly by employing the kernel trick. There is a large variety of so-called *graph kernels* (GKs) to do this. GKs can be used in combination with any kernel method, typically a SVM, to solve the GC/GR problem. While there is a large variety of different GKs, we will focus on those that are based on the previously described family of WL algorithms.

WL subtree kernel

WL shortest path kernel

Higher dimensional WL kernels

2.3.3 Graph Neural Networks

Spatial GNNs

Spectral GNNs

3 Learning to Aggregate on Graphs

4 Evaluation

5 Conclusion

5.1 Review

5.2 Future Directions

Appendix

A

Bibliography

[AB73] George W. Adamson and Judith A. Bush. „A method for the automatic classification of chemical structures“. In: Information Storage and Retrieval 9.10 (1973), pp. 561–568 (cit. on p. 10).

[Ban+18b] Priyanka Banerjee, Andreas O. Eckert, Anna K. Schrey, and Robert Preissner. „ProTox-II: a webserver for the prediction of toxicity of chemicals“. In: Nucleic Acids Research 46.W1 (2018), W257–W263 (cit. on p. 10).

[Drw+14] Malgorzata N. Drwal, Priyanka Banerjee, Mathias Dunkel, Martin R. Wettig, and Robert Preissner. „ProTox: a web server for the in silico prediction of rodent oral toxicity“. In: Nucleic Acids Research 42.W1 (2014), W53–W58 (cit. on p. 10).

[GL16] Aditya Grover and Jure Leskovec. „node2vec: Scalable Feature Learning for Networks“. In: (July 3, 2016). arXiv: 1607.00653v1 [cs.SI] (cit. on p. 11).

[LM14] Quoc Le and Tomas Mikolov. „Distributed Representations of Sentences and Documents“. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. ICML'14. Beijing, China: JMLR.org, 2014, pp. II–1188–II–1196. arXiv: 1405.4053v2 [cs.CL] (cit. on p. 12).

[Lue+18] Thomas Luechtefeld, Dan Marsh, Craig Rowlands, and Thomas Hartung. „Machine Learning of Toxicological Big Data Enables Read-Across Structure Activity Relationships (RASAR) Outperforming Animal Test Reproducibility“. In: Toxicological Sciences 165.1 (2018), pp. 198–212 (cit. on p. 10).

[MH16] Vitalik Melnikov and Eyke Hüllermeier. „Learning to Aggregate Using Uninorms“. In: Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, 2016, pp. 756–771 (cit. on pp. 3, 4).

[MH19] Vitalik Melnikov and Eyke Hüllermeier. „Learning to Aggregate: Tackling the Aggregation/Disaggregation Problem for OWA“. In: Proceedings of The Eleventh Asian Conference on Machine Learning. Ed. by Wee Sun Lee and Taiji Suzuki. Vol. 101. Proceedings of Machine Learning Research. Nagoya, Japan: PMLR, 2019, pp. 1110–1125 (cit. on p. 5).

[Mik+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. „Distributed Representations of Words and Phrases and Their Compositionality“. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119. arXiv: 1310.4546v1 [cs.CL] (cit. on p. 10).

[Nar+17] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, et al. „graph2vec: Learning Distributed Representations of Graphs“. In: (July 17, 2017). arXiv: 1707.05005v1 [cs.AI] (cit. on p. 12).

[Per+14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. „DeepWalk: online learning of social representations“. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14. ACM Press, 2014. arXiv: 1403.6652v2 [cs.SI] (cit. on p. 11).

[WL68] Boris Weisfeiler and Andrei A. Lehman. „A reduction of a graph to a canonical form and an algebra arising during this reduction“. In: Nauchno-Technicheskaya Informatsia 2.9 (1968), pp. 12–16 (cit. on p. 6).

[WW86] Peter Willett and Vivienne Winterman. „A Comparison of Some Measures for the Determination of Inter-Molecular Structural Similarity Measures of Inter-Molecular Structural Similarity“. In: Quantitative Structure-Activity Relationships 5.1 (1986), pp. 18–25 (cit. on p. 10).

Websites

[Ban+18a] Priyanka Banerjee, Robert Preissner, Andreas Eckert, and Anna K. Schrey. ProTox-II - Prediction Of Toxicity Of Chemicals. Charité – Universitätsmedizin Berlin. 2018. URL: http://tox.charite.de/protox_II/ (visited on Nov. 18, 2019) (cit. on p. 10).

[Tes] Toxicity Estimation Software Tool (TEST). EPA. URL: https://www.epa.gov/chemical-research/toxicity-estimation-software-tool-test (visited on Nov. 18, 2019) (cit. on p. 10).

[Tox] ToxTrack - Cheminformatics Modeling. ToxTrack Inc. URL: https://toxtrack.com/ (visited on Nov. 18, 2019) (cit. on p. 10).

List of Figures

Erklärung zur Masterarbeit

Ich, Clemens Damke (Matrikel-Nr. 7011488), versichere, dass ich die Masterarbeit mit dem Thema *Learning to Aggregate on Structured Data* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinn nach entnommen habe, wurden in jedem Fall unter Angabe der Quellen der Entlehnung kenntlich gemacht. Das Gleiche gilt auch für Tabellen, Skizzen, Zeichnungen, bildliche Darstellungen usw. Die Masterarbeit habe ich nicht, auch nicht auszugsweise, für eine andere abgeschlossene Prüfung angefertigt. Auf § 63 Abs. 5 HZG wird hingewiesen.

Paderborn, 16. Februar 2020

Clemens Damke