Relational Learning approaches for Recommender Systems

Giovanni Pellegrini

Advisor: Andrea Passerini

Department of Information Engineering and Computer Science
University of Trento

Ph.D. Dissertation

University of Trento September 2021

Abstract

Learning on relational data is a relevant task in the machine learning community. Extracting information from structured data is a non-trivial task due to the combinatorial complexity of the domain and the necessity to construct methods that work on collections of values of different sizes rather than fixed representations. Relational data can naturally be interpreted as graphs, a class of flexible and expressive structures that can model data from diverse domains, from biology to social interactions. Graphs have been used in a huge variety of contexts, such as molecular modelling, social networks, image processing and recommendation systems. In this manuscript, we tackle some challenges in learning on relational data by developing new learning methodologies. Specifically, in our first contribution, we introduce a new class of metrics for relational data based on relational features extraction technique called *Type Extension Trees* [60]. This class of metrics defines the (dis)similarity of two nodes in a graph by exploiting the nested structure of their relational neighborhood at different depth steps. In our second contribution, we developed a new strategy to collect the information of multisets of data values by introducing a new framework of learnable aggregators called *Learning Aggregation Functions*. We provide a detailed description of the methodologies and an extensive experimental evaluation on synthetic and real world data to assess the expressiveness of the proposed models. A particular focus is given to the application of these methods to the recommendation systems domain, exploring the combination of the proposed methods with recent techniques developed for *Constructive Preference Elicitation* [42] and *Group Recommendation* tasks.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Since the middle of the '90s, thanks to the development of the internet and its life-changing disruption in many aspects of society, the amount of data produced every day is exponentially increasing. The International Data Corporation esteems that the total amount of production and consumption of data has reached 59 zettabytes (ZB) in 2020[1], and in the next three years, the amount of data produced in the last 30 years will be doubled. Consequently, strategies to store, retrieve and process this huge amount of information has raised interest in both the industry and academia. Although it is estimated that the vast majority of the data produced is unstructured, in many cases, the pieces of information produced relate to each other, possibly forming very complex networks. For instance social networks [44], images [46], biological structures [110], road networks [1], recommendation systems [108] and web pages [129] are domains that present the same underlying structure, having as basic components the entities involved and the various relationships that connect them. The general way to represent the data of this domains is to model it as a *graph*. Graphs are data structures used to represent objects (nodes) and their connections or relationships (edges), are extremely flexible and generalize other architectures such as *Relational Databases* [32] and *Knowledge Bases* [73]. Every data source in which its entities or base components present interconnections can be represented as a graph structure. Learning representation on graphs is a challenging task that has been investigated by the artificial intelligence and machine learning communities for the last twenty years. While early works on learning on graphs focused on the construction of kernel functions to compare

[1] https://www.idc.com/getdoc.jsp?containerId=prUS46286020

structures and attributes [10, 16, 18, 21], the increasing developments in neural networks architectures lead to the recent advances in deep learning techniques for graphs [98, 119, 70, 52, 127] that, together with other techniques for processing non-euclidean data, recently culminated in the new field denominated *Geometric Deep Learning* [19]. Despite the notable amount of research work on graph data, we aim to provide new solutions for determining the similarity of graphs' nodes and overcome some limitations when aggregating information from multisets of data and, in the case of graphs, neighbors nodes. Another consequence of the increasing amount of information produced on the web is the necessity to develop methods that are able to match and retrieve contents based on some query or specific necessities of a system or a user. Recommendation systems are nowadays a well established technology that have risen interests in the last decade. Applications in areas such as e-commerce [99], video streaming services [2] and social networks [78], to mention a few, use state-of-the-art techniques in real world applications. Part of this work discuss how such systems can take advantage of relational structures to enhance their expressiveness and overcome some problems in this domain.

1.1 Contributions

This thesis embodies my effort to study and research machine learning application on graph structured data. Another aspect of my PhD work is given by the possible application of this techniques into recommendation systems. The main contributions of this thesis can be summarized in three part:

- Definition of a new class of metrics on relational data;
- A solution for learning to aggregate information from multisets of data;
- Extension of some recommendation techniques to integrate the newly proposed techniques.

The following manuscript is the result of the following papers published in peer reviewed conferences and journals

- Jaeger, M., Lippi, M., Pellegrini, G., and Passerini, A. (2019). Counts-of-counts similarity for prediction and search in relational data. *Data Mining and Knowledge Discovery*, pages 1–44

- Dragone, P., Pellegrini, G., Vescovi, M., Tentori, K., and Passerini, A. (2018a). No more ready-made deals: constructive recommendation for telco service bundling. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 163–171. ACM

and the following manuscript accepted at IJCAI '21:

- Pellegrini, G., Tibo, A., Frasconi, P., Passerini, A., and Jaeger, M. (2020). Learning aggregation functions. *arXiv preprint arXiv:2012.08482*

1.2 Outline of the Thesis

This manuscript is structured as follows:

Chapter 2: this chapter summarizes the literature on machine learning techniques for graph data. Here we introduce some of the most relevant research works that constitute the foundation of my research;

Chapter 3: in this chapter we present the first contribution of this work, i.e. a class of metrics that define the similarity between the entities in a relational graph. Initially we illustrate how to construct the relational features upon which the metric is applied, developing later a whole set of distance function that use this features. Then we show how to modify some of the proposed metrics in order to perform metric learning tasks. We provide experimental analysis for classification, regression and information retrieval tasks.

Chapter 4: in this chapter we illustrate a framework to learn aggregation functions for multisets of values. We show the learning procedure for this class of aggregators and provide extensive experimental evaluation on many benchmarks;

Chapter 5: here we discuss the extension of some recommendation systems to integrate the methodologies proposed in the Chapters 4 and 5.

Chapter 6: finally, we draw some conclusions and discuss some future research directions.

Chapter 2

Background

2.1 First Order Logic

In propositional logic, the world is composed of *facts* which do not present any internal structure, and can be either *true* or *false*. *First Order Logic* (FOL) can be interpreted as an extension of propositional logic, in which the world is composed of *objects* and these objects are linked by some *relations*. First Order Logic is particularly useful to model real world scenarios in which entities characteristics, relationships with other entities, and the action taken by the entities influence the state of the world. Another way to think about FOL, with some restrictions, are some type of knowledge bases as, for instance, relational databases. FOL can perfectly represent relational databases if the right components are provided in the formulation of the particular world described in the database. In this section, we introduce the concepts and notation to define the FOL formalism. The FOL language is composed of four sets of symbols: *variables*, *constants*, *predicates* and *functions*. Constant symbols represent the instance of the objects in the domain, e.g. `moon`, `mars`, `jupyter`. Variable symbols denote sets of objects (e.g. `Planets`). Both constants and variables can be typed, so objects of a particular type belong to the variable of the same type, and typed variables represent only objects of the same type. Function symbols operate on tuples of objects and return as output an object (e.g. `Satellites(earth)`). Predicates symbols are used to describe the relationships among (tuple of) objects, or attributes of the objects (e.g. `OrbitsAround(europa, jupyter)` or `HasRing(saturn)`). For clarity, throughout the manuscript we use capital letters (X, Y, Z) to denote variables, and lower

case letters (x, y, z) to denote constants. Bold letters represent multiple occurrences (arities) of variables or constant symbols. A *term* is any constant, any variable and any function symbol with terms as arguments. A single predicate that takes as input a tuple of terms is called an *atom*. FOL shares with propositional logic the logical *connectives* such as the negation $(\neg)$, the conjunction $(\wedge)$, the disjunction $(\vee)$, the single and double implication $(\Rightarrow, \Leftrightarrow)$. In addition to propositional logic, FOL introduces two new symbols: the *universal quantifier* $(\forall)$, which is true if a formula is satisfied for all the object in a specific domain, and the *existential quantifier*$(\exists)$, which is true if at least one of object in the domain satisfies the formula. A *formula* is a composition of one or more atoms and logical connectives. A formula is said to be *propositional* if no quantifiers appears in it. A *Ground term* is a term containing no variables and a *ground formula* is a formula with ground terms as arguments. A world is represented by a *relational signature*, a set which contains all the variables, constant, predicates and function symbols. Finally, a *Herbrand interpretation* assigns a truth value to every ground formula obtainable from the world.

2.2 Graphs

In this section we provide a formal definition of a graph and the notation that will be used throughout the entire manuscript. A graph is a tuple of objects $G = (V, E)$, where V is the set of nodes (or vertices) and E is the set of edges connecting the nodes. $|V|$ represents the total number of nodes in the graph. We denote with lower case letter the instance objects of the respective sets, i.e. $v \in V$ denotes a node, and $(v_i, v_j) \in E$ denotes the edge connecting the node v_i to the node v_j. For the edges we will also use the notation $e_{i,j} \in E$ when convenient. A node can also contain a self-loop, i.e., $(v_i, v_i) \in E$. A graph is *undirected* if $(v_i, v_j) \Leftrightarrow (v_j, v_i)$, otherwise the graph is *directed*. If a graph is weighted, the edges are associated with a weight $w_{i,j} > 0$ determining the "strength" of the connection. In weighted undirected graphs $w_{i,j} = w_{j,i}$. A compact representation of the connections is given by the the *adjacency matrix* $\boldsymbol{A} \in \mathbb{R}^{|V| \times |V|}$ in which $\boldsymbol{A}_{i,j} = 1$ if there exists an edge (v_i, v_j). In weighted graphs $\boldsymbol{A}_{i,j} = w_{i,j}$. The neighbourhood $\mathcal{N}(v)$ of a node v is defined as the set of nodes to which v is linked to by one or more edges, i.e. $\mathcal{N}(v_i) = \{v_j | (v_j, v_i) \in E\}$. A graph is *labeled* if nodes (and possibly edges) are associated with a symbol

from an alphabet Σ. We denote with L_v the label of a node and with $L_{e_{i,j}}$ or $L_{(v_i,v_j)}$ the label of an edge. The *shortest-path* between two nodes is the minimal sequence of nodes and edges traversed to connect two non-neighbor vertices. A *random walk* starting at node v_i is a sequence of nodes $v_{i,1}, v_{i,2}, \ldots$ obtained according to the probability $P(i_{k+1}|i_1, \ldots, i_k) = \boldsymbol{A}_{i_k,i_{k+1}}$. We use the symbol $\boldsymbol{x} \in \mathbb{R}^n$ to represent a n-dimensional feature vector attached to a node, referred to as the *state* of the node. The matrix $\boldsymbol{X} \in \mathbb{R}^{|V|\times n}$ groups together all the nodes' states. If all the edges of the graph belong to a single class, i.e., they all represent one type of connection, then we call the graph *single-relational*. Otherwise, two nodes can possibly be connected by multiple edges belonging to different classes, in this case we talk about *multi-relational* graphs.

2.3 Relational Features

Statistical Relational Learning (SRL) is a branch of machine learning that studies the models that can be constructed upon relational data. A relational knowledge base can be expressed in the form of a graph or a network, resulting in a multi-relational data structure, in which the entities of the relational domain are represented by the nodes and the relationships between entities correspond to the edges. Notable research work in this field can be found in [92, 59, 38]. Most reinforcement learning or deep learning techniques construct their model from data which present an attribute-value representation, in which an entity (or data point) is characterized by some *features* expressed as a fixed size vector (or matrix) of values. A meaningful representation can be determined by manually setting the size of the vector by the system expert, or by obtaining a latent representation with any representation learning technique. The clear advantage of the vectorial representation is given by the geometrical meaning of the data points, allowing for models to take advantage of linear algebra tools to perform transformations and operations of the vectors in an euclidean space, learning how to separate or fit the data through some hyperplanes. This non-relational representation assumes that all the information of the entities is *flattened* into a single table where all data points are *independent and identically distributed* (*i.i.d.*). This is one of the strongest assumption made by non-relational models. Contrarily, SRL models take advantage of the relational structure to define features for entities allowing for a greater flexibility and generalization abil-

ity by removing the *i.i.d.* assumption. Another notable distinction is that, in non-relational models, heterogeneous entities must be represented by the same vectorial structure to be processed by a learning model. For instance, in *collaborative filtering* recommendation the vector associated to a user and the vector representing an object must have the same number of dimensions in order to perform the dot product. Meanwhile, relational features can be constructed upon heterogeneous structures of entities and not being bounded to have a fixed representation. However, designing meaningful feature representations for relational data is a relevant problem due to the combinatorial nature of the relational information that can be exploited from the domain. In this section we describe a technique to create features for entities in a relational domain, and how machine learning models can be constructed upon this features. The research works presented in Chapter 3 is developed upon a specific formalism of this type, called *Type Extension Trees* (TETs). We show that learning models of different nature can be developed upon the values extracted with TETs, such as logistic regression models and relational similarity measures. The rest of this section describes the motivation behind TETs and the formal rules to create such features.

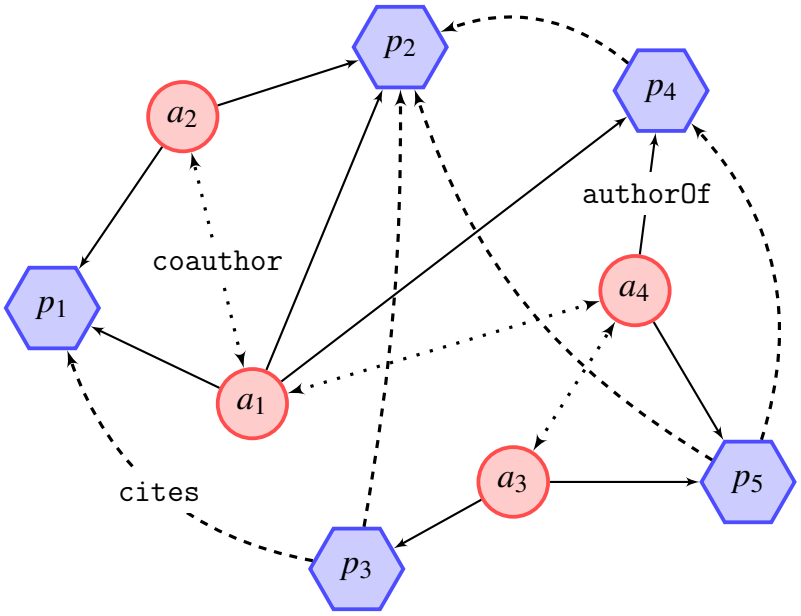
2.3.1 Type Extension Trees

An object in a relational domain is characterized by the values of its attributes, the objects in its relational neighborhood, and the attributes of such objects. Inductively, each object in the set of neighbors is characterized by its own set of neighbors' objects and their attributes. Therefore, by considering the different depths of neighbors, in a relational domain the supply of information is possibly unlimited. When analyzing the neighbors structure of a multi-relational graph, one can also restrict the relational neighborhood to a specific type of relationship or keep or filter out neighbors that present certain characteristics. Moreover, it is possible to define "chains" of relational patterns, and characterize an object as a combination of these patterns. Many SRL models lack a precise definition of the relational features needed to train the learning model, resulting in an oversimplified representation of the data (e.g., propositionalization approaches) or a tight integration of the data representation and the learning model [14, 117]. Therefore, defining a clear feature space and representation language is a fundamental step to clearly separate the relational features

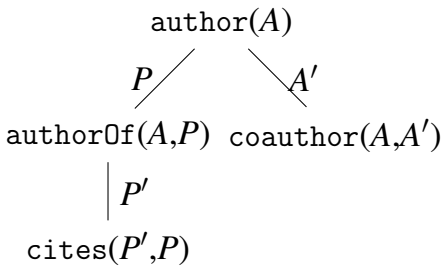
value space and the learning model applied over them. Type Extension Trees fill this gap between data representation and learning models by introducing a formalism that allows for the use of First Order Logic to construct nested combinatorial features while giving a sound definition of the features value space. TETs do not only focus on simple numerical values of the attributes of the object but on the statistics of nested structures of neighbors objects that satisfy a relational constraint, to create what is called a *counts-of-counts feature*. The main concept of a TET feature is to count the occurrences of similar sub-structures at different depths of the relational neighborhood, for a single object or a set of objects, while allowing for defining relational constraints on the neighbors relationships. Some commonly used measures in information retrieval can be expressed as counts-of-counts features, for instance the *h-index* and *tf-idf* index of documents [60]. TETs are defined with a formalism based on FOL, making them easy to encompass features for both single relation and multi-relational data. Initially introduced in [58], several research studies have shown the possible application of TET in citation networks and forecasting prediction [75, 60]. In this section, we review the definition and semantics of the TET formalism, which is used as the fundamental building block for describing the new methodologies introduced in Chapter 3.

Definition 1 *Consider a relational signature R of relational symbols of different arities. An R-literal is an atom $r(\boldsymbol{V})$ such that $r \in R \cup \{=\}, \boldsymbol{V} = V_1, \ldots, V_k$ is a tuple of variable symbols. An R-type is a conjunction of R-literals. A Type Extension Tree is a tree structure in which nodes are labeled with R-types and edges can be labeled with variables.*

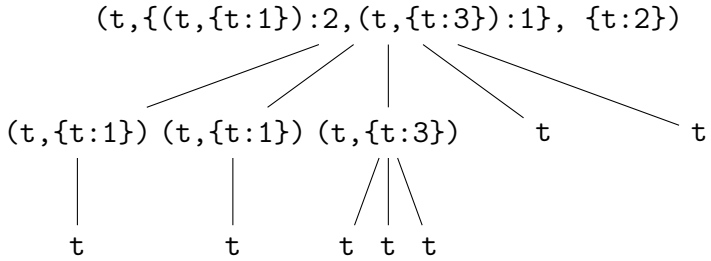
Throughout the manuscript, we will use the terms TET and TET-feature to denote the same concept. Each node of the TET is assigned with a specific *type* determined by the literals specified in the node. Labeled edges are related to universal quantifiers in FOL, in that they bind all the occurrences of the variables specified in it to the subtree pointed by it. An example of TET-feature is given in Figure 2.1b, defined for the relational domain in Figure 2.1a. Here the TET defines a feature for an author entity. The variables that appear in the TET but that are not quantified in any edge are called *free* variables, determining the variables for which a TET defines a feature. If the edges are not labeled, no variable of the TET is quantified; therefore, the TET is equivalent to a propositional representation of the variables. We call it *propositional* TET if no edge



a) Relational graph of authors a and papers p.



b) TET-feature defined on the relational signature of a)



c) TET-value for the author a_1

Fig. 2.1 Graphical representation of: a) a relational graph, b) a TET and c) a TET-value. The TET b) defines the feature structure for an author entity. To obtain the feature values for objects the root variables are instantiated and the TET extracts the count-of-count features from the relational graph. The TET-value c) of the author a_1 uses only *true* values.

is quantified by any label, in that it represents a feature for a specific set of objects. A formal representation of a TET-feature is the following:

$$T(\boldsymbol{V}) = [\alpha(\boldsymbol{V}), (\boldsymbol{Y}_1, T_1(\boldsymbol{V}, \boldsymbol{Y}_1)), \ldots, (\boldsymbol{Y}_m, T_m(\boldsymbol{V}, \boldsymbol{Y}_m))]$$

$T(\boldsymbol{V})$ indicates a TET defined for the free variables in $\boldsymbol{V} = V_1, ..., V_k$. The symbol $\alpha(\boldsymbol{V})$ represents the root of the TET, while $(\boldsymbol{Y}_k, T_k(\boldsymbol{V}, \boldsymbol{Y}_k)), k = 1, \ldots, m$ is the k_{th} subtree, with an edge from the root to the child node labeled with variables $\boldsymbol{Y}_k$.

Example 1 *The formal representation of the TET-feature in Figure 2.1b is*

$$\begin{aligned} T(A) = [&\texttt{author}(A), \\ &(P, [\texttt{authorOf}(A,P), \\ &\qquad (P', [\texttt{cites}(P',P)])]), \\ &(A', [\texttt{coauthor}(A,A')])] \end{aligned}$$

The TET above defines a relational feature for a single variable. The root literal determines whether the input objects of the TET is an author. Variable A is the only free variable of the TET since it is not quantified in any edge of the tree. The two children of the root represent two sub-TETs rooted at the literals $\texttt{authorOf}(A,P)$ *and* $\texttt{coauthor}(A,A')$*, each linked through edges labeled with variables* P' *and* A' *respectively. Following the same formal structure, the two sub-TETs are*

$$\begin{aligned} T_1(A,P) &= [\texttt{authorOf}(A,P), (P', [\texttt{cites}(P',P)])] \\ T_2(A,A') &= [\texttt{coauthor}(A,A')] \end{aligned}$$

T_1 *defines a feature for a pair of author-paper, while* T_2 *defines a feature for an author-author pair pf objects. Recursively,* T_1 *is linked to a child node, the sub-TET given by*

$$T_{1,1}(P,P') = [\texttt{cites}(P',P)]$$

Notice that each sub-TET defines a TET per-se defined for its own set of free variables.

We use the notation $T(\boldsymbol{V})$ to indicate a TET which defines a feature for the free variables $\boldsymbol{V} = V_1, ..., V_k$, but other free variables can be introduced in the substructure of the TET. For an assignment $\boldsymbol{v} = v_i, ..., v_k$, $T(\boldsymbol{V})$, the a *TET-value* $V(T(\boldsymbol{v}))$ is defined as follows:

Definition 2 *Let $T(\boldsymbol{V})$ be a TET with free variables $\boldsymbol{V} = V_1, ..., V_k$, $\boldsymbol{v} = v_1, ..., v_k$ a k-tuple of entities and $\alpha(\boldsymbol{V})$ the atom corresponding to the root of T. Given an interpretation function $I : \alpha(\boldsymbol{v}) \rightarrow \{t, f\}$, the TET-value $V(T(\boldsymbol{v}))$ for an assignement $\boldsymbol{v}$ is inductively defined as follow:*

- ***Base step:***
  $T(\boldsymbol{V}) = [\alpha(\boldsymbol{V})]$, *then*
  - *if $I(\alpha(\boldsymbol{v})) = f$ then $V(T(\boldsymbol{v})) = f$*
  - *if $I(\alpha(\boldsymbol{v})) = t$ then $V(T(\boldsymbol{v})) = t$*
- ***Inductive step:***
  $T(\boldsymbol{V}) = [\alpha(\boldsymbol{V}), (\boldsymbol{Y}_1, T_1(\boldsymbol{V}, \boldsymbol{Y}_1)), \ldots, (\boldsymbol{Y}_m, T_m(\boldsymbol{V}, \boldsymbol{Y}_m))]$, *then*
  - *if $I(\alpha(\boldsymbol{v})) = f$ then $V(T(\boldsymbol{v})) = f$*
  - *if $I(\alpha(\boldsymbol{v})) = t$ then*

$$V(T(\boldsymbol{v})) = (t, \mu(\boldsymbol{v}, \boldsymbol{Y}_1, T_1), \ldots, \mu(\boldsymbol{v}, \boldsymbol{Y}_m, T_m))$$

where $\mu(\boldsymbol{v}, \boldsymbol{Y}_i, T_i)$ is a multiset of TET-values.

For the sake of synthesis, we use t, f for *true* and *false* evaluations of I and the symbol γ to denote a TET-value. A multiset $\mu(\boldsymbol{v}, \boldsymbol{Y}_i, T_i)$ contains all the TET-values of the sub-TET with T_i as root, each value obtained by evaluating the literal of T_i with instances $\boldsymbol{v}$ and all the instances of the objects in $\boldsymbol{Y}_i$. Since TETs model counts-of-counts features, we denote with $\{\gamma_1 : n_1, ..., \gamma_l : n_l\}$ a multiset containing n_j copies of the TET-value γ_j.

Example 2 *Consider the relational domain and TET-feature showed in Figure 2.1a,b respectively. We want to extract the TET-value for the author a_1. By Definition 2, the value $V(T(a_1)) \equiv \gamma(a_1)$ becomes*

$$\gamma(a_1) = (t, \{f : 2, (t, \{f : 4, t : 1\}) : 2, (t, \{f : 2, t : 3\}) : 1\}, \{f : 2, t : 2\})$$

Counting the number of *false* and *true* evaluations allow to define statistics based on frequency, such as the fraction of f or t values with respect to the total number of counts, which can be used for instance to define a simple predictive model [60]. However, one can consider only the positive evaluations of the TET-value. Considering the t values only is a valid strategy to construct the feature, in that by the definition of TET-value if the interpretation function evaluates the literal of an intermediate (non-leaf) node as *false* the inductive step is no longer developed, therefore "pruning" branches of the TET-value. A graphical representation of the TET-value showed in Example 2 containing only *true* evaluations is illustrated in Figure 2.1b.

2.4 Kernel Methods on Graphs

Defining similarity measures between graphs is among the most investigated tasks on this type of structures over the last twenty years. Later in this manuscript we will propose a new class of distance functions that measure the dissimilarity between tuples of graph entities, therefore we revise here some well known techniques presented in the literature to define similarities, i.e. *kernel functions* for graphs. We start by defining some early techniques, to arrive to recent deep learning techniques to learn representations on graphs.

Support Vector Machines (SVMs) [17, 34] are among the most popular classes of methods in machine learning. Initially developed during the '90s, they dominated the machine learning field for years thanks to their versatility, efficiency, simplicity, and soundness. SVMs have been applied to solve many different tasks, from character recognition to computational biology. There are several reasons why SVMs became popular: the optimization problem is convex, therefore, an optimal solution can always be found; they limit the number of calculation steps required to classify new examples, as comparisons with all data points is not needed, but just some vectors, called *support vectors*, are useful to determine the final prediction. Finally, SVMs can be applied to solve classification and regression tasks. SVMs belong to a class of machine learning techniques called *kernel methods* [21]. The following definition of a positive definite kernel is originally provided in [101]:

Definition 3 *A symmetric function $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ which for all $m \in \mathbb{N}$, $x_i \in \mathscr{X}$ gives rise to a positive definite Gram matrix, i.e. for which for all $c_i \in \mathbb{R}$*

we have

$$\sum_{i,j=1}^{m} c_i c_j K_{ij} \geq 0, \text{ where } K_{ij} := k(x_i, x_j)$$

is called a positive definite kernel.

A kernel function is used as a substitution of the dot product. A kernel is a function that acts as a norm and abstracts the concept of the dot product to non-Hilbert spaces.

Definition 4 *Consider a set of data objects x_i belonging to an arbitrary space $\mathscr{X}$. Let k be a function that takes in input pairs of objects and returns a positive scalar value, i.e. $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}_{\geq 0}$. Assume a mapping function $\phi : \mathscr{X} \to \mathbb{R}^d$ that projects the object into a Hilbert space H_d exists, then $k(x_i, x_j) =< \phi(x_i), \phi(x_j) >$ is a valid kernel.*

Note that the input space $\mathscr{X}$ can be any space, including Euclidean spaces. The kernel application allows for what is referred to as the "kernel trick" [101]. The transformation $\phi(x)$ that maps an object into a vector (usually in a high dimensional space) can be expensive to compute. The advantage is that the kernel computes the similarity directly on the objects input space without passing through the additional step of the transformation. Linear models usually consist of finding a hyperplane that can efficiently separate two regions of the space or interpolate the data points, minimizing some error measure. In most scenarios, the data is not linearly separable, i.e., no hyperplane can perfectly separate two classes of points without misclassification. Thanks to kernels, it is possible to transform the data into a higher dimensional space, becoming linearly separable, allowing a hyperplane (in the transformed space) to precisely divide the data. The following are some examples of kernels [47] for objects belonging to some vector space $x \in \mathbb{R}^n$:

- Linear: $k(x_i, x_j) =< x_i, x_j >$
- Polinomial: $k(x_i, x_j) = (1 + < x_i, x_j >)^p$, $p \in \mathbb{N}$
- Gaussian: $k(x_i, x_j) = exp\big(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\big)$, $\sigma \in \mathbb{R}$

Moreover, kernel methods have been investigated to deal with complex data structures such as trees [82, 131], sequences [10] and graphs [106, 62]. The following sections of the chapter will provide a more detailed review of graph

kernels and in general machine learning methods for learning graph representation.

2.4.1 Weisfeiler-Lehman Graph Kernel

The problem of determining whether two graphs present an identical structure is called the *graph isomorphism problem* (or graph isomorphism test). Two graphs are isomorphic if there exists a bijection between the vertex of the first graph and the vertex of the second graph. The two are identical if, for every node of one graph, there is a unique node of the other graph that presents the very same connections. This is a computationally hard problem, in fact it is not known if there exists an algorithm that can solve it in polynomial time, nor to be NP-complete [49]. However, beyond the complexity of the problem, using the isomorphism test as a similarity measure would be of limited practical use. The test result is a binary value: true if the two graphs are identical, false otherwise, instead of expressing a degree of similarity between the two. Moreover, complex structures present characteristics that makes it more difficult to define such measures. When designing a similarity measure, or a kernel for graphs, the main issue is the permutation invariance of the function, i.e., the order of the input objects should not change the final output. In this section we illustrate some state-of-the-art kernel methods that take advantage of different concepts to formalize permutation invariant functions for graphs. Most of them are based on the concept of isomorphism test introduced by Weisfeiler and Lehman in 1968 [123]. The first algorithm we introduce is referred to in the literature as the 1-dimensional Weisfeiler-Lehman (1-WL) test, also called *naive vertex refinement* or *color refinement* algorithm. The color refinement idea is to assign a label (color) to each node of the graph, and iteratively update the labels by using the nodes neighbors' label as local information. Eventually, the labels will terminate in a stable state (i.e., the labels distribution does not change from the previous iteration), or it terminates when the number of iterations corresponds to the number of nodes in the graph. Note that if the two graphs have a different number of nodes, the test fails, therefore, to start the iterative process the two graphs are required to have the same number of nodes. The algorithm operates as follows. Consider a graph $G = (V, E, L)$ and a labeling function $l : \Sigma^* \to \Sigma$ that calculates the labels of the nodes. Σ is the alphabet of the labels, $L_v \in \Sigma$. At the initial step $i = 0$, the same label is as-

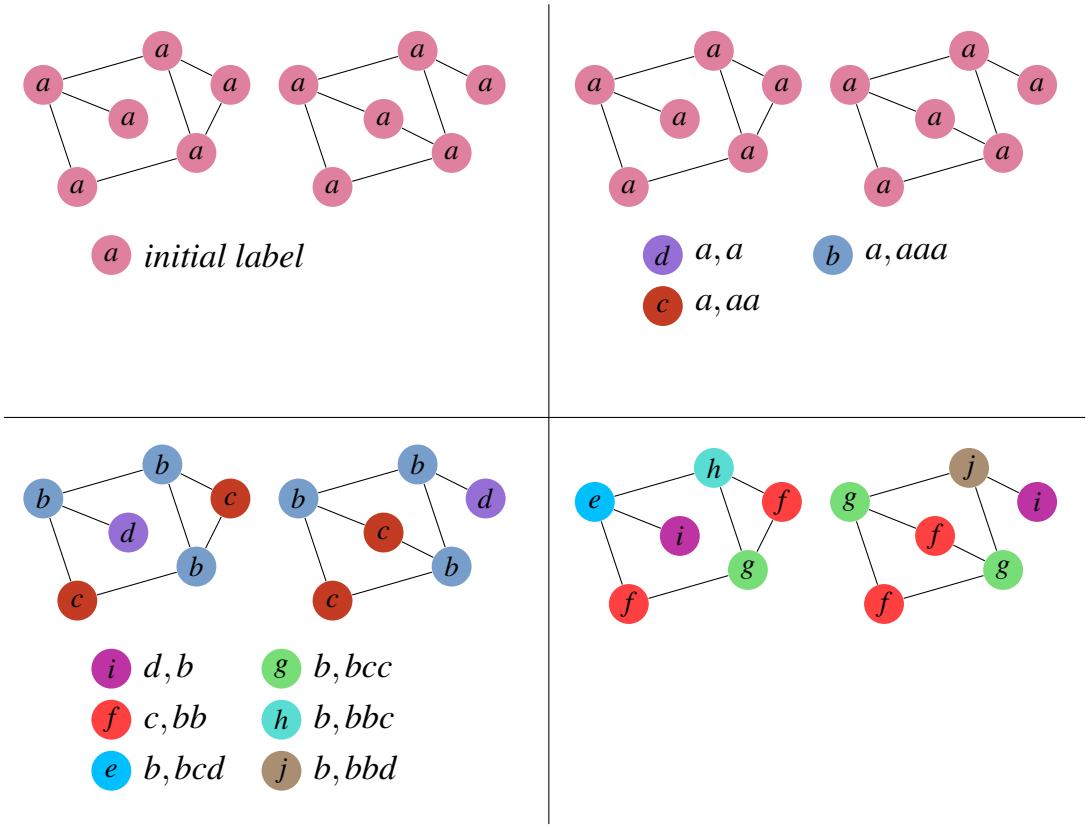


Fig. 2.2 Example of the iterations of the 1-dimensional Weisfeiler-Lehman algorithm. Each quadrant shows the initial state of the labels at the beginning of the iteration, and the relabeling step considering the different combinations of nodes' label and neighbors multiset. The algorithm stops at iteration 3: the coloring of the nodes of the graphs is different.

signed to each node. Then, iteratively, each node is assigned with the multiset of labels of its neighbors at the previous step. At the final step, the labeling function produces the new labels for the nodes by considering the concatenation of the node label and the multiset of the neighbors label. The relabelling step is effective only if the ordering function of the multiset and the labeling functions are injective, i.e., the nodes labels are the same if they share the same labels and all the neighbors labels. However, an injective function on multisets is required to be permutation invariant, i.e., the elements position does not influence the final result. One can alleviate this requirement by enforcing some ordering of the multiset elements and then applying the labeling. We denote as $h(\cdot)$ the function that operates on the multiset. Two graphs have an identical structure if the multisets of the nodes' label are the same under some *canonical representation*, usually implemented as a partial ordering. Examples of canoni-

cal representation can be obtained using a lexicographical ordering or counting the repetition of identical ordered labels. The complete labeling operation for one node is performed as follows:

$$L_v^i = l(L_v^{i-1}, h(\{L_u^{i-1} | u \in \mathcal{N}(v)\})) \tag{2.1}$$

At each iteration, this procedure yields two multisets of labels, $\{L_v^{i^*} | v \in V\}$ and $\{L_{v'}^{j^*} | v' \in V'\}$, one for each graph, which are used to determine the isomorphism. The algorithm terminates when the canonical representations of the multisets are different, or at most at step $i^* = |V|$. If the two graphs have different representations, it can be concluded that they are not isomorphic; on the contrary, if the representation is identical, it is not possible to state that they are isomorphic. Although the test does not guarantee isomorphism when it succeeds, in practice it is a valid test for the majority of graphs [4]. An example of the test is illustrated in Figure 2.2. Each iteration of the test yields a labeling of the nodes $G_i = (V, E, L_i)$ based on the labels at the previous iteration. The concept behind general *Weisfeiler-Lehman Graph Kernel* (WLGK) kernel is to utilize the representations of the two graphs obtained at each step j to measure the similarity between them. Consider a positive semidefinite kernel on graphs $k(G, G')$, which we call base kernel; a WLGK kernel can be constructed as the weighted summation of the base kernel function for an arbitrary number of steps h:

$$k_{WL}^h(G, G') = \sum_{i=0}^{h} w_i k(G_i, G_i') \tag{2.2}$$

where $w_i \geq 0$ can be every weight choice (e.g., a discount factor to assign different importance to further steps of the algorithm). Note that being $k_{WL}^h(G, G')$ a linear combination of positive semidefinite kernels, it retains the characteristic of being a positive semidefinite kernel. This framework for defining WLGK kernels can assume any arbitrary number of iterations that can be adjusted on fixed statistics of the problem, e.g., the graphs size, or can be learned or determined through some heuristic. However, the WLGK kernel limit as described above is that it can only handle discrete alphabets of symbols for labeling the nodes and not continuous assignments. Extensions that overcome this problem are later discussed in Section 2.4.3.

2.4.2 WL Subtree Kernel

A novel implementation of the WLGK framework, named *Weisfeiler-Lehman subtree kernel* [106], is defined over a feature mapping of the graphs states: at each step i of 2.1, a function maps the label of the nodes to a new feature representation and concatenates all the representations of the previous step $k < i$ with the map of i. The base kernel can be seen as the simple dot product on this new feature maps. The mapping, as described in [106], works as follows. Consider the multiset $\Sigma_i = \{\sigma_0^i, \ldots, \sigma_{|\Sigma_i|}^i\}$ composed of the labels of the nodes at step i with $\Sigma_i \subseteq \Sigma$. The function $c_i(G, \sigma_k^i)$ counts the number of occurrences of the symbol σ_k, $k = 1, \ldots, |\Sigma_i|$, of the multiset Σ_i. The feature for a graph at iteration j is computed by concatenating the counts of all label symbols at each iteration step up to j:

$$\phi_{WL}^j(G) = (c_0(G, \sigma_1^0), \ldots, c_0(G, \sigma_{|\Sigma_0|}^0), \ldots, c_j(G, \sigma_1^j), \ldots, c_j(G, \sigma_{|\Sigma_j|}^j)) \quad (2.3)$$

The WL subtree kernel is obtained by computing the dot product of the feature mapping of the two graphs at each iteration of the general WL kernel:

$$k_{WLsubtree}^j(G, G') = < \phi_{WL}^j(G), \phi_{WL}^j(G') > \quad (2.4)$$

The $k_{WLsubtree}$ allows for a fine-grained comparison of the graphs obtained with efficient time. Moreover, it is equivalent to the k_{WL} kernel if only the feature mapping for one iteration is considered and the base kernel is a Dirac kernel [106] returning 1 if the two mappings are the same, zero otherwise.

Example 3 *Consider the two graphs in Example 2.2 and a number of iteration $j = 2$. The feature mappings of the two graphs with the counting function $c_i(G, \sigma_k^i)$ are:*

$$\begin{aligned}
\phi_{WL}^2(G) &= (c_1(G,a), c_2(G,b), c_2(G,c), c_2(G,d)) \\
&= (6,3,2,1) \\
\phi_{WL}^2(G') &= (c_1(G',a), c_2(G',b), c_2(G',c), c_2(G',d)) \\
&= (6,3,2,1)
\end{aligned}$$

and the $k_{WLsubtree}$ kernel (Equation 2.4) gives

$$k^2_{WLsubtree} =< \phi^j_{WL}(G), \phi^j_{WL}(G') >= 50$$

2.4.3 WL Kernels with continuous labels

A practical limitation of the methods described in the previous sections is that they work only with a symbolic representation of the node and their connections, relying on discrete attribute labels to describe them. Kernels based on sub-pattern similarities identify two identical nodes only if their local structure and neighbors' labels match exactly, and do not consider more expressive attribute representations that characterize the nodes. In many real-world tasks, the nodes of the graphs represent objects associated with attribute-value vectors with values possibly ranging in discrete and continuous domains. For instance, in a social network graph, the users are described in terms of personal interests, the number of posts, and activeness; chemical structures have chemical and physical properties; the intensity of traffic characterizes edges in road networks. The underlying concept between graph kernels that handle continuous attributes is to define the similarity based on both structural similarities of the local neighborhood and comparison of the attributes of the nodes and edges. A common strategy to define structure similarity is to decompose the graph into substructures or patterns that are then matched together via one (or more) base kernels, and the kernel(s) evaluations are summed together. Kernels for graphs that follow this scheme lie under the category of $\mathscr{R}$*-convolutional* kernel. The underlying kernel computation of this type follows the scheme

$$k_w(v) = \sum_{v \in V(G)} \sum_{v' \in V(G')} k_w(v, v') k_v(v, v') \tag{2.5}$$

where $k_w(v, v')$ determines the weight of the connection between the nodes (if the edges of the graph are weighted, one can replace k_w with the edge weight) and $k_v(v, v')$ measure the similarity between the attributes of the nodes. The *GraphHopper* [45] and *GraphInvariantKernel* [85] methods use this scheme to define kernels on continuous attributes. GraphHopper defines a kernel over families of shortest paths as the summation of *path kernels*. The base kernel considers only paths of the same length. A path is defined as the ordered sequence of nodes to traverse in the graph from a starting vertex to an ending one,

i.e., $\pi = [v_1, \cdots v_k]$. Here $k_w(v, v')$ counts the number of times v and v' appear at the same coordinate, or *hop*, of paths π and π'. In GraphHopper, the $k_w(v, v')$ kernel represents the designer's only variable choice, being it free of learnable parameters. A more general approach is given in [85] where authors define the weighting kernel as the combination of an *vertex invariant kernel* $k_{inv}(v, v')$ and a rescaling factor based on nodes local neighborhood. The invariant function assigns a higher value to two nodes when they appear in the same substructure having the same structural role, allowing for a smoother similarity comparison than hard matching function and avoiding unwanted correlation induced by the structural ordering of the vertices.
Other kernels use different strategies to handle a continuous range of values. Recent work proposed an approach named *Waisserstein Weisfeiler-Lehman Graph Kernels* (WWLGK) [115] that uses the *earth-mover's distance* (EMD) to compare the distributions induced by the nodes representations. In a few words, the earth-mover's distance measure the (dis)similarity between two distributions by calculating the cost of transforming one distribution into the other. More details on EMD are provided in Section 3.4. In WWLGK, the node's attributes are encoded into embeddings via a graph embedding function $g : G \rightarrow \mathbb{R}^{|V| \times m}$ where m is the number of dimensions of the embeddings. The output matrix X_G contains the encoding of node v_i at its i^{th} row. The earth mover's distance is then applied to the two graphs' embeddings, essentially treating the two matrices as the probability distribution over the nodes. As in the WL kernel scheme, the embeddings of the nodes can be refined through an iterative process, and the similarity of the graphs measured at each refinement step. To obtain a new vector for a node, the vectors of the neighbors are averaged together and summed to the target node representation. As shown in further sections, the averaging operation is much similar to the propagation scheme used in *Graph Neural Networks*, where the function for node aggregation plays a fundamental role in defining the expressiveness of this methods.

2.4.4 Walks and paths kernels

As illustrated in the previous section, graph kernels that compare the structural similarity of parts of the graphs are referred to as R-convolutional kernels. This methods require the pattern structure to be specified in advance, or the size of the subgraph to consider, usually relying on small size structures to be compu-

tationally efficient. An alternative to this class of methods is to compare the sequences of vertex and edge labels encounter when traversing the graph. This sequences can be defined as the shortest path between pairs of nodes, or as random walks. Although computing all the shortest paths in a graph is a NP-Hard problem, a shortest path can be computed in polynomial time. The *shortest-paths kernel* introduced by Borgwardt and Kriegel [16] decomposes the graphs into shortest paths between pairs of nodes, comparing the length of the paths and the labels of the nodes traversed. The first step is to decompose a graph $G(V,E)$ into the shortest-paths between all pair of nodes, using an algorithm such as Dijkstra or Floyd-Warshall. The result of the decomposition is a graph $S(V,E_s)$ containing the same set of vertices of the original graph and a new set of edges E_s. An edge of $e \in E_s$ connects two nodes v_i and v_j in S if there is a path connecting them in G, and it is labeled with the distance (number of hops) of the path.

Definition 5 *Let $G = (V,E), G' = (V',E')$ be two graphs, and $S = (V,E_s), S' = (V',E'_s)$ be the shortest-path decomposition of G, G' respectively. A shortest-path kernel is defined as*

$$k_{sp}(G,G') = \sum_{e \in E_s} \sum_{e' \in E'_s} k_{path}(e,e') \tag{2.6}$$

if k_{path} is positive definite, then k_{sp} is also a positive definite kernel.

$k_{path}(e,e')$ determines a kernel that compares the labels of the starting and ending nodes of the path and the shortest distance between them. Let $e = (u,v)$, $e' = (u'v')$. Let also L_u, L_v be the labels of the nodes L_e, L'_e be the labels of the edges. The $k_{path}(e,e')$ kernel can be formulated as

$$k_{path}(e,e') = k_l(L_u,L'_u) \cdot k_l(L_v,L'_v) \cdot k_d(L_e,L'_e) \tag{2.7}$$

where k_l is a kernel for comparing nodes' labels and k_d is a kernel comparing the edges' labels (i.e., the length of the paths). The complexity of the shortest-path kernel is $O(n^4)$. Another strategy to decompose the graph into sequence patterns is to perform *random walks*. A random walk is obtained starting from a node and iteratively pick a node in the neighborhood and repeat the process for the selected node. Similarly to the shortest paths setting, the feature space is composed of the labels of nodes and the edges. Although the underlying idea is

similar to the shortest path method, i.e., to compare the labels of the nodes and labels present in sequences, the random walk adopts another approach based on the *direct product graph*.

Definition 6 *The direct product of two labeled graphs $G = (V,E)$ and $G' = (V',E')$, denoted as $G \times G' = (\mathscr{V},\mathscr{E})$, is defined as follows:*

$$\mathscr{V} = \{(v,v') \in V \times V' | L_v = L'_v\}$$
$$\mathscr{E} = \{((u,u'),(v,v')) \in \mathscr{V} | (u,v) \in E \wedge (u',v') \in E' \wedge L_{(u,v)} = L_{(u',v')}\}$$

A vertex (edge) in $G \times G'$ has the same label as the corresponding vertex (edge) in G and G'.

The following definition defines the *direct product kernel*

Definition 7 *Let G, G' be two graphs. $\boldsymbol{A}_\times$ and $\mathscr{V}_\times$ denote the adjacency matrix and the vertices of the direct product graph $G \times G'$. Given a sequence of weights $\lambda = (\lambda_1, \lambda_2, \ldots) | \lambda_t \in \mathbb{R}, \lambda_t \geq 0, \forall t \in \mathbb{N}$, the direct product kernel is defined as*

$$k_{rw}(G,G') = \sum_{i,j=1}^{|\mathscr{V}|} \Big[\sum_{l=0}^{\infty} \lambda_l \boldsymbol{A}_\times^l \Big]_{i,j} \tag{2.8}$$

To ensure the convergence of the algorithm the values $\lambda_t < \frac{1}{\lambda_\times}$ where $\lambda_\times$ denotes the largest eigenvalue of the matrix $\boldsymbol{A}_\times$. The advantage of using the direct product graph is that a random walk in $G \times G'$ corresponds to take simultaneously random walks on G and G'. However, the computational time to create the direct product graph is $O(n^6)$, making this method applicable only to small scale graphs.

2.5 Deep Learning for Graphs

Neural architectures that operate on graphs take inspiration from recent *Deep Learning* architectures to define new models for graph data. In the last decade, the contributions to this field have seen a remarkable improvement. The motivation behind this increment in the research activity is given by recent developments in the field of Deep Learning and to overcome the limitations imposed by some previous methods as kernels. As with kernel methods, *Graph Neural Networks* (GNNs) can learn nodes representations to embed a graph via

a mapping function, but instead of relying on a fixed mapping (as defined in kernel methods), GNNs aims to apply a learnable function for producing the mapping. This section focuses on GNNs techniques for supervised classification on graphs, which are usually employed to solve three types of tasks: node, edge, and graph classification. GNNs have adopted and adapted many different concepts commonly used in Deep Learning, for instance, taking inspiration from recursive architectures (RecGNNs) and convolutions strategies (ConvGNNs), but also from mechanisms like attention [118] and variational inference [69]. Early investigations implement recursive techniques to graph data, starting at the middle of the '00s with the works by Gori et al. [51] and Scarselli et al. [98]. Since the introduction of this first neural network models, the development of neural architectures to process graph data has dramatically increased in the last years [52, 67, 70, 119, 127], reaching state-of-the-art performances in applications like protein-protein interaction [52, 119], movie domains [127], internet domains [52, 127] and citation networks [52, 70, 119]. Due to the incredible amount of research work published during the last years, researchers questioned how powerful GNNs models are and their representation abilitie [127, 12]. In the Graph Neural Networks (GNNs) context, a node or an edge of the graph is associated with a feature vector $h \in \mathbb{R}^n$ that represents its state. The states of the components can be used to classify the nodes, the edges, and the classification of the entire graph. We provide an overview of some representative GNNs models.

2.5.1 The Graph Neural Network Model

The *Graph Neural Network Model* [98] firstly introduced the application of neural networks to handle general graph structures, e.g., acyclic, cyclic, directed, and undirected graphs. Graph Neural Network Model defines a class of methods that recurrently update the state of the nodes by applying a parametrizable function to the local structure until a stable configuration for all the nodes is reached. The model defines a *local transition function*, which, for a given node, takes in input several local structural information and return a new state h, and a *local output function* responsible for the final prediction given the state of a node. To extend the model to graph prediction tasks, the authors also define a *global transition function* and a *global output function* that instead of being applied to a single node take in input all the nodes' state of the graph.

The GNN Model provides a general yet flexible framework due to its ability to incorporate a high volume of information into its local transition function, including neighbors and edges labels and states. Let L_v be the label of the observed node v, $L_{\mathscr{C}(v)}$ the labels of the edges connected to v, $\boldsymbol{h}_{\mathscr{N}(v)}$ and $L_{\mathscr{N}(v)}$ the neighbors states and labels respectively. The local transition function and the output function are defined as follows:

$$\begin{aligned} \boldsymbol{h}_v^t &= f_{\boldsymbol{w}}(L_v, L_{\mathscr{C}(v)}, \boldsymbol{h}_{\mathscr{N}(v)}^{t-1}, L_{\mathscr{N}(v)}) \\ \boldsymbol{o}_v^t &= g_{\boldsymbol{w}}(\boldsymbol{h}_v^t, L_v) \end{aligned} \tag{2.9}$$

where the functions $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$ can be implemented as neural networks with arbitrary layers, hence parametrized and fully derivable. The functions can be extended in several ways, for instance, the set of neighbors can include nodes that are n steps away to the observed node; additional structural information (as the direction of edges in a directed graph) can be encoded in the form of a matrix and added as input to f_w; alternatively to using a single f_w for all the nodes, one can assign a specific function to each node. To ensure convergence, the local transition function is required to be a contraction map with respect to the node state, i.e. $\exists \mu, 0 \leq \mu < 1 \; s.t. \; \|f_w(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq \mu \|\boldsymbol{x} - \boldsymbol{y}\|$ where $\|\cdot\|$ is a vector norm.

2.5.2 Graph Convolutional Networks

Graph Convolutional Network (GCN) [70] is a spectral based GNN that applies the concept of convolution in CNNs to graph structures. The initial state of the nodes is passed through several convolutional layers where learnable filters determine the transformation of the states. In this setting, the resulting states from the layer l are stacked into a matrix $\boldsymbol{H}^l \in \mathbb{R}^{|V| \times d}$, $\boldsymbol{W}^l$ represents the filter and $\boldsymbol{A}$ is the adjacency matrix of the graph. The convolution operation at each layer is defined as

$$\boldsymbol{H}^{l+1} = \sigma(\boldsymbol{A}\boldsymbol{H}^l\boldsymbol{W}^l) \tag{2.10}$$

where $\sigma(\cdot)$ is a non-linear function. The adjacency matrix $\boldsymbol{A}$ allows for summing only over the neighbors' nodes and not the entire set of nodes of the graph. If the formulation of the convolutional layer would use the plain adjacency matrix, two problems arise; 1) since the matrix is not normalized, nodes with a

huge number of nodes will have high values while nodes with the restricted neighborhood will have low values representing the state; 2)only the states of the neighbors would be accounted in the transformation, not the state of the node itself. To overcome this problems, the authors propose two solutions: the first is to add self loops to each node, i.e. summing the identity matrix $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$. The second solution is to normalize $\boldsymbol{A}$ by the diagonal matrix $\hat{\boldsymbol{D}}$ which encodes the node degree so that the rows of $\hat{\boldsymbol{A}}$ sum up to 1. Rewriting Equation 2.10 we obtain

$$\boldsymbol{H}^{l+1} = \sigma(\hat{\boldsymbol{D}}^{-\frac{1}{2}}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{l}\boldsymbol{W}^{l}) \tag{2.11}$$

By adding the self-loops to the adjacency matrix, GCN convolution is directly linked to the hash function used in the WL isomorphism test. However, the convolution operation should learn an injective function to behave as the hash in the WL test, but no injectivity guarantees are provided.

2.5.3 GraphSAGE

GraphSAGE is a type of spatial, graph convolutional network meant for inductive learning representation of a graph's nodes. The algorithm generates node embeddings by sampling the neighbors' nodes and then learning an aggregation function that combines the neighbors' states into a single vector representation. Then, the state of the observed node is concatenated to the aggregated representation, and the resulting vector is passed through a non-linear transformation. This sampling and aggregation process takes place for an arbitrary level of exploration K, for which it is defined the number of nodes to sample s^k, a trainable aggregation function Aggr^k, a weight matrix $\boldsymbol{W}^k$ and a nonlinearity function σ. Algorithm 1 illustrates the steps for learning to generate the embeddings.

Although there is no limitation for GraphSAGE to process all the nodes, the sampling of the neighbors allows for computation on large scale graphs, which might represent a limitation for other methods that use all the nodes of the graph. Being created for inductive learning representation, GraphSAGE also has the advantage of being applied to unseen nodes in the graph (i.e., inductive setting) once the embedding functions are learned. A crucial aspect in GraphSAGE is given by the choice of the aggregation functions used in

Algorithm 1 GraphSAGE embedding generation algorithm

Input: Graph $G(V,E)$; input states $\{x_v, \forall v \in V\}$; weight matrices $\boldsymbol{W}^k, k = 1, \cdots, K$; aggregation functions $Aggr^k, k = 1, \cdots, K$; non linear function σ;

Output: Node embeddings $\boldsymbol{z}_v, \forall vinV$

$\boldsymbol{h}_v^0 \leftarrow \boldsymbol{x}_v, \forall v \in V$
for $k = 1, \cdots, K$ **do**
    for $v \in V$ **do**
        $\boldsymbol{h}_{\mathcal{N}(v)}^k \leftarrow \text{Aggr}^k(\boldsymbol{h}_u^{k-1}, \forall u \in \mathcal{N}(v))$
        $\boldsymbol{h}_v^k \leftarrow \sigma(\boldsymbol{W}^k \cdot \text{Concat}(\boldsymbol{h}_v^{k-1}, \boldsymbol{h}_{\mathcal{N}(v)}^k))$
    $\boldsymbol{h}_v^k \leftarrow \boldsymbol{h}_v^k / \|\boldsymbol{h}_v^k\|_2, \forall v \in V$
$\boldsymbol{z}_v \leftarrow \boldsymbol{h}_v^K, \forall v \in V$

the architecture. Usually, GNNs models rely on the *sum* as the aggregation function to condense the information of the neighbors, but this represents a limitation since other functions like the *max*, *mean* and even more complex statistics could leverage the representation ability of this models. Therefore, GraphSAGE has been tested with three different types of aggregation function: a *mean aggregator*, which computes the average element-wise of the neighbors' states, an *LSTM aggregator* and a *Pooling aggregator* that computes the max-pooling of the neighbors. The LSTM can not be considered a reliable aggregation function in that it is not permutation invariant; however, to provide more stability to invariance, a random permutation is applied to the input set of neighbors.

2.5.4 Graph Isomorphism Networks

The authors in the original publication [127] address the question of what is the representational power of the GNNs models, proving that they are *at most* as expressive as the WL isomorphism test. Specifically, they demonstrate that for two non-isomorphic graphs G and G', if a GNN $A : G \rightarrow \mathbb{R}^d$ maps the two graphs with different embeddings, also the WL isomorphism test decides they are not isomorphic. By adapting the labeling function of the WL test (Equation 2.1), we can obtain a general formulation for GNN as follows:

$$\boldsymbol{h}_v^k = \phi(\boldsymbol{h}_v^{k-1}, f(\{\boldsymbol{h}_u^{k-1} | u \in \mathcal{N}(v)\}) \tag{2.12}$$

Therefore, for a GNN to be as expressive as the WL test, it is necessary for the functions ϕ and f to be injective. Note that the function f processes the multi-set of the neighbors' states; thus, it has to be injective and permutation invariant to the set elements. A recent research work shows how to construct a permutation invariant function to process a set of data using a *sum-decomposition* strategy. The resulting framework for processing this data is called *DeepSets* [128]. The sum-decomposition theorem states that

Theorem 1 *A function $f(X)$, with input a set X defined over a countable domain, is invariant to the permutation of instances $\{x_1, \cdots, x_n\}$ of X if and only if it can be decomposed in the form $\rho(\sum_{x \in X} \varphi(x))$, for suitable transformations ρ and φ.*

It is easy to note the correlation of Theorem 1 with the hash function of the WL test, originally define also for a countable domain of labels. However, in practice, the node vector components $\boldsymbol{h}_v$ are defined over the domain of reals, i.e., $\boldsymbol{h}_v \in \mathbb{R}^d$, therefore Theorem 1 has minimal use in real-world applications. The authors of [128] conjecture that the theorem also holds for uncountable domains, however, they could only prove the theorem holds for sets of *fixed* size. For constructing an invariant function for GNNs the sum-decomposition theorem has been extended to multisets of data. To construct a GNN with the sum decomposition properties, the functions ρ and φ are defined as an MLP given that it can express compositions of function, in this case, $\rho \circ \varphi$. Xu et al. [127] define a GNN model called *Graph Isomorphism Network* that satisfy the conditions of Equation 2.12 as

$$\boldsymbol{h}_v^k = \text{MLP}^k\Big((1+\varepsilon^k) \cdot \boldsymbol{h}_v^{k-1} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{h}_u^{k-1}\Big) \tag{2.13}$$

where ε represents a parameter that can be fixed or learned during the process.

Chapter 3

Distance Measures Based On TETs

3.1 Relational Data Structures

Machine learning models are often constructed on top of the data structure they process. Perhaps the most common representation of data in machine learning is that of attribute-value structure, in which data objects are characterized by a finite set of characteristics that belong to a particular domain. This representation gives the advantage to represent the data as a vector, i.e., as a point in an n-dimensional Euclidean space. Such representation gives the advantage of working with linear algebra tools, in which space transformations and hyperplane separation are the basic concepts for many established methods since the rise of modern machine learning (e.g., SVM and perceptron). However, despite the powerful mathematical tools used to process this data, the attribute-value representation lacks a structural dependence among its elements. It is the model's job to discover this dependency, not an intrinsic characteristic of the data itself. As we have seen in Chapter 2, FOL is a tool that instead models and captures this dependency in a well-defined mathematical formalism. For this reason, the terminology *relational data* usually refers to FOL or structures that can be represented in FOL. For instance, relational databases can be expressed in First Order Logic without any loss of generality, in which the tables of the database represent the template (variables) for instance objects, which present specific characteristics, and the connection between tables map to the relations

or predicates. These structural dependencies are very strongly correlated with graphs or networks; in fact, FOL entities and relations can be represented as an equivalent *multi-relational* graph. This type of graph differs in the diverse type of nodes and connections that link the nodes. In the classical representation of a graph, the nodes are homogeneously represented as instances of one class, and the connections represent a very specific relation happening between the objects. For instance, in a road network, the nodes are the intersection of the roads, and the roads themselves are the edges of the graph. When representing molecules as graphs, the nodes are the chemical elements and the edges are the chemical bonds. In a multi-relational graph, there exists more than one node type, and two nodes can have multiple connections, each connection potentially belonging to a different type. For instance, in a social network, the nodes could be both users and "pages", linked together by actions as the "like" or "follow" representing the edges.

3.2 Logistic Evaluation Function

The evaluation of a TET $T(\boldsymbol{V})$ as $V(T(\boldsymbol{v}))$ as defined in Section 2.3.1 leads to a complex nested multiset structure of Boolean values. Jaeger et al. [60] presented two approaches for using these values as the basis for prediction tasks and similarity measures:

- by defining a *discriminant function* f that maps TET values $V(T(\boldsymbol{v}))$ to real numbers, and that can be used for binary classification tasks;
- by defining a metric $d(V(T(\boldsymbol{v})), V(T(\boldsymbol{v}')))$ on TET values that can be used for distance-based methods such as nearest-neighbor prediction.

These approaches have been tested on the artificial task of predicting the binary attribute for an author having an *h-index* greater than 7. Although the h-index is a deterministic function of the data, its combinatorial nature and discontinuity makes it non-trivial to be learned with perfect precision. Again, in [60] the authors present the task of assigning a negative label to authors with h-index < 7 and a positive one to h-index > 7. They report an F1 scores of 61.3% and 91.2% for this prediction task when using the discriminant function and nearest-neighbor prediction, respectively. These results indicate that the discriminant function and metric definition of [60] are not flexible and powerful enough to fully exploit the information given by a TET value $V(T(a))$ to learn how to solve the $(h > 7)$ prediction task with 100% accuracy. For this reason, we developed a rich class of evaluation functions $l^{\boldsymbol{\beta}}$ that map TET values to real numbers and that are parameterized by an adjustable vector $\boldsymbol{\beta}$. Additionally, we define a metric on the nested multiset structure of real numbers that is generated by the recursive evaluation of $l^{\boldsymbol{\beta}}$ on a TET value $V(T(\boldsymbol{v}))$. The evaluation function $l^{\boldsymbol{\beta}}$ will be used in two different models as the discriminant function for direct class prediction, and also as the basis for defining metrics $d^{\boldsymbol{\beta}}$ that can represent specific similarity concepts by adjusting the parameters $\boldsymbol{\beta}$ of the underlying function.

We first define a parameterization of a TET:

Definition 8 *Let $T(\boldsymbol{V})$ be a TET. A* weight assignment $\boldsymbol{\beta}$ *for T assigns a non-negative real number to all non-leaf nodes and all edges of T. A weight assignment can be written as $(\beta_0^r, \beta_1^r, \ldots, \beta_m^r, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m)$, where β_0^r is the weight*

assigned to the root, β_i^r is the weight assigned to the edge from the root to its ith child, and $\boldsymbol{\beta}_i$ is the weight assignment to the ith sub-tree.

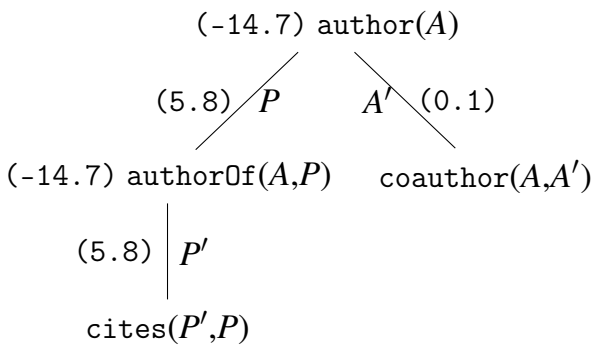


Fig. 3.1 TET-feature for an author a. The numbers attached to the nodes and the edges represent the weights of a possible parameter assignment $\boldsymbol{\beta}$.

Given a weight assignment for TET, we define a function on TET values γ via a recursive definition over the nested multiset structure of γ:

Definition 9 *For a TET T with weight assignment $\boldsymbol{\beta}$ the* logistic evaluation *function $l^{\boldsymbol{\beta}}$ is defined as follows. Let $\gamma = V(T(\boldsymbol{v}))$ be a value.*

- ***Base step:***
  - *If $\gamma = f$, define $l^{\boldsymbol{\beta}}(\gamma) := 0$.*
  - *If $\gamma = t$, define $l^{\boldsymbol{\beta}}(\gamma) := 1$.*
- ***Inductive step:***
  - *If $\gamma = (t, \mu_1, \ldots, \mu_m)$, with multisets μ_i of values of sub-trees T_i, define:*

$$l^{\boldsymbol{\beta}}(\gamma) := \sigma\left(\beta_0^r + \sum_{i=1}^{m} \beta_i^r \sum_{\gamma' \in \mu_i} l^{\boldsymbol{\beta}_i}(\gamma')\right)$$

where σ is the sigmoid function $\sigma(x) = 1/(1+e^{-x})$.

In other words, the evaluations of the values belonging to the multiset of the sub-TET T_i, i.e., $\gamma' \in \mu_i$, are weighted by the parameter β_i^r. Then, the values computed for all sub-TETs are summed together, with the root parameter β_0^r acting as the bias weight. For the rest of the chapter we employ the sigmoid

function as non-linear activation, in that for the definition of the distance function d^{β} we need the values to be in $[0,1]$, but to use the evaluation function as predictor there are no limitations on the choice of the activation function. By storing the execution outputs of the function at each node, the recursive evaluation of a TET value γ leads to a nested multiset structure of real numbers that matches the same structure of γ. We refer to this new structure of logistic evaluation values as the *logistic evaluation tree*, denoted $L^{\boldsymbol{\beta}}(\gamma)$.

Example 4 *In this example we give a demonstration of the applicability of the logistic evaluation function to determine the binary class of an author having an h-index > 3. Consider the TET $T(A)$ as shown in Figure 2.1 (b). The root and the left sub-TET represents a sufficient feature to represent the h-index of an author, therefore for computing this task we could eliminate the contribution of the right branch and still be able to recover the h-index. We introduce a parametrization $\boldsymbol{\beta}$ for the TET $T(A)$ as $\boldsymbol{\beta} = (-14.7, 5.8, 0.1, (-14.7, 5.8))$ in the notation of Definition 8. We apply the evaluation function to the TET value $V(T(a_1))$, starting from the leaf nodes up to the root. The evaluation of each leaf node representing a* true *value returns a 1. The leaf nodes labeled with* false *which are omitted in the figure all evaluate to 0. Now consider the sub-value $(t, \{t : 3\})$ of the TET node $\texttt{authorOf}(A,P)$ at the middle level of Figure 2.1 (b). We obtain:*

$$l^{(-14.7,5.8)}((t,\{t:3\})) = \sigma(-14.7 + 5.8 \cdot (1+1+1)) = 0.937$$

The omitted false *values have no impact on this calculation, since they would only add a number of 0 terms to the inner sum. Similarly, $l^{(-14.7,5.8)}((t,\{t : 1\}) = 0.0001$. Note that here we have $m = 1$, since the underlying TET has no branching. Now, for the top-level we have to combine the evaluations of two branches. From the evaluation of the root we then obtain:*

$$\begin{aligned} l^{\boldsymbol{\beta}}(\gamma) &= \sigma(-14.7 + 5.8 \cdot (0.93 + 0.0001 + 0.0001) + 0.1 \cdot (1+1)) \\ &= 0.0001 \end{aligned}$$

Figure 3.2 (middle) shows the logistic evaluation tree $L^{\boldsymbol{\beta}}(\gamma)$ induced by this computation of $l^{\boldsymbol{\beta}}(\gamma)$. The final value $l^{\boldsymbol{\beta}}(\gamma)$ is the root of the tree.

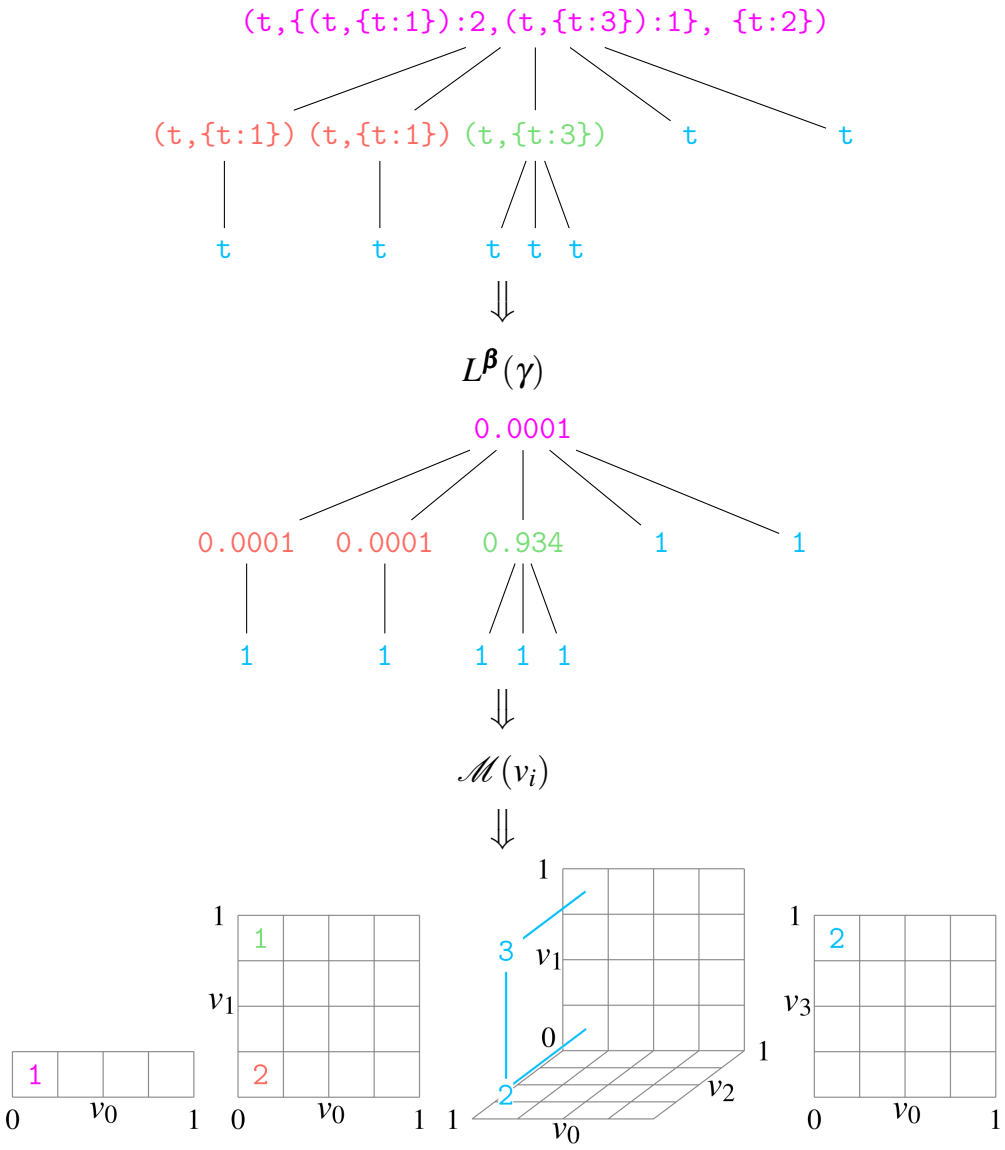


Fig. 3.2 Graphical representation of the steps from a TET-value to its histograms approximation. The value γ is computed by the logistic evaluation function which outputs the logistic evaluation tree $L^{\beta}(\gamma)$. Then the multi-value paths $\mathscr{M}(v_i)$ are constructed from $L^{\beta}(\gamma)$. The histograms are computed for nodes (from left to right) v_0, v_1, v_2, v_3 with $N = 4$. Empty cells correspond to zero count. A node v_i with depth d_i in the TET is represented by a d_i-dimensional histogram whose dimensions correspond to the TET nodes on the path from the root to v_i. Since all paths start with the same value (0.0001) in the first component, and end with a 1 in the last component, only a single one-dimensional slice in the 3-dimensional histogram is populated with nonzero counts.

3.2.1 Neural network perspective and weight learning

In the preceding examples we have computed the recursively defined $l^{\boldsymbol{\beta}}(\gamma)$ by an inductive bottom-up propagation of sub-values. These computations are similar to the forward propagation in a neural network with a sigmoid activation functions. Indeed, one can think of γ as defining a neural network structure, and $\boldsymbol{\beta}$ as defining a neural network weights and biases. A given TET T with logistic evaluation parameters $\boldsymbol{\beta}$ then can be seen as a template for the construction of example-specific neural networks, and the TET formalism represents a highly flexible and expressive way to define such templates. Under this neural network perspective, the logistic evaluation tree in Figure 3.2b shows the activations of the neural networks defined by the structure a) for the parametrization $\boldsymbol{\beta}$. Note that here the inputs to the neural network are always equal to 1 at all input (leaf) nodes. Given a supervised learning objective expressed by a differentiable loss function on the logistic evaluation values $l^{\boldsymbol{\beta}}(\gamma)$, one can apply standard backpropagation rules to compute the gradient of the loss for a single example $\boldsymbol{v}$ with respect to $\boldsymbol{\beta}$, and learn $\boldsymbol{\beta}$ using any of the many available (stochastic) gradient descent techniques.

3.3 Histogram Approximation

The nested multiset structures of $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}))$ gives a detailed description of $\boldsymbol{v}$ in terms of quantitative features of its relational neighborhood, as defined by T and the parameters $\boldsymbol{\beta}$ of the logistic evaluation function. We now aim to use this description as a basis for defining distances between entities $\boldsymbol{v}, \boldsymbol{v}'$. In large and highly connected graphs, the full structure $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}))$ will become very large, and not suitable to support fast distance computations, or to store pre-computed $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}))$ for all $\boldsymbol{v}$. We therefore introduce an approximation of $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}))$ by a collection of multi-dimensional histograms. The approximation will be constant in size for all $\boldsymbol{v}$, and independent of the size of the data graph.

As the first step towards this approximation, we approximate the full tree structure of $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}))$ by multisets of value paths.

Definition 10 *Let v be a node in the TET T, and $r = v_0, v_1, \ldots, v_{k-1}, v_k = v$ the path leading from the root r of T to v. Let $\gamma = V(T(\boldsymbol{v}))$ be the value of T for entities $\boldsymbol{v}$. A sequence $\gamma_0, \ldots, \gamma_k$ is a* value path for *v in γ, if*

- $\gamma_0 = \gamma$, *and* $\gamma_i \neq f$ *for all i.*
- *For* $i < k$:
  if v_{i+1} *is the* $j(i)$*th child of* v_i *in* T, *and* $\gamma_i = (\mu_{i,1}, \ldots, \mu_{i,j(i)}, \ldots, \mu_{i,m_i})$,
  then $\gamma_{i+1} \in \mu_{i,j(i)}$ *with multiplicity* $k_{i+1} \geq 1$.

The multiset of value paths for v *is the multiset that contains the value path* $\gamma_0, \ldots, \gamma_k$ *with multiplicity* $\prod_{i=0}^{k-1} k_{i+1}$.

Example 5 *For the TET in Figure 3.1 (top) let the be* $v_0 =$ `author`, $v_1 =$ `authorOf`, $v_2 =$ `cites` *and* $v_3 =$ `coauthor`. *Then the value* γ *shown in Figure 3.2 (top) contains for* v_2 *the value path*

$$(t, \{(t, \{t:1\}):2, (t, \{t:3\}):1\}, \{t:2\}), (t, \{t:3\}), t$$

with multiplicity 3, since the root have by definition a multiplicity of 1, multiplied by the multiplicity of the sub-value $(t, \{t:3\})$ *which is 1, multiplied by the multiplicity of the true values at the leaf, i.e. 3.*

Let $\boldsymbol{\beta}$ be a parameter vector for the logistic evaluation function for T. A sequence of real numbers $t_1, \ldots, t_k$ is a *logistic value path for* v if there exists a value path $\gamma_0, \ldots, \gamma_k$ for v such that $t_i = l^{\boldsymbol{\beta}}(\gamma_i)$. The *multiset of logistic value paths for* v, denoted $\mathscr{M}(v)$ is the multiset that contains the logistic value path $t_1, \ldots, t_k$ with a multiplicity equal to the sum of multiplicities of value paths $\gamma_0, \ldots, \gamma_k$ that induce $t_1, \ldots, t_k$.

Example 6 *Consider the nodes* v_0, v_1, v_2, v_3 *as in Example 5. Then for the logistic evaluation tree in Figure 3.2 (middle):*

$$\begin{aligned}
\mathscr{M}(v_0) &= \{(0.0001):1\} \\
\mathscr{M}(v_1) &= \{(0.0001, 0.934):1, (0.0001, 0.0001):2\} \\
\mathscr{M}(v_2) &= \{(0.0001, 0.934, 1):3, (0.0001, 0.0001, 1):2\} \\
\mathscr{M}(v_3) &= \{(0.0001, 1):2\}
\end{aligned}$$

We observe that while the value paths of a node v_{i+1} in some sense extend the value paths of its parent v_i, it is not the case that from $\mathscr{M}(v_{i+1})$ the multiset $\mathscr{M}(v_i)$ can be constructed. The values of the logistic evaluation function lie in the interval $[0,1]$. The elements of a logistic value path multiset $\mathscr{M}(v)$ are

vectors of a fixed length equal to the depth d of v in T (defining the depth of the root to be 1), and hence are elements of $[0,1]^d$. We partition the interval $[0,1]$ into N equal-width bins, leading to a discretization of $[0,1]^d$ into N^d cells. This leads to an approximate representation of $\mathscr{M}(v)$ by a d-dimensional histogram, which we call the *node histogram h* for v. Arranging the node histograms for all nodes of T in a tree structure isomorphic to T leads to the *node histogram tree* (NHT) as an approximation for the logistic value tree $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v})))$. The number of bins N is a hyperparameter that balances the accuracy of the values' representation and compactness for computational efficiency.

Example 7 *Figure 3.2 provides an overview of the steps from a TET-value γ to its histogram representation. Firstly, the logistic evaluation function computes the values that compose $L^{\boldsymbol{\beta}}(\gamma)$. The logistic multi value paths are constructed from the logistic values, and then, for a chosen value of N, the histograms are created. The counts in the histograms represent the distribution of the logistic values for different nodes in the TET, starting from the root (which contains a single value) to the leafs. In the Example shown in Figure 3.2 the value of the root is close to zero, hence its multiplicity (1) falls in the first bin. The second histogram shows the 2-dimensional distribution of the values of v_1. The values are distributed along the dimension of the root node, showing that 2 papers obtained values close to 0 and 1 paper with a value close to 1. Going down to the node v_2 the values are distributed along the new dimension in the bin closest to 1, being v_2 a leaf node, therefore, having values equal to 1. The multiplicities at this level are multiplied by the multiplicities at the previous node, so the single citation is multiplied by the number of papers that have that single citation (2), whereas there is a single paper with 3 citations, therefore the multiplicity in this case is 3. The last histogram is computed for the node v_3.*

3.4 NHT Metrics

We now proceed to define a metric on NHTs. There can obviously be many different approaches for defining such a metric, and in the following we make a number of design choices. We will not be able to prove for each such choice that it is the only possible or optimal one. However, they all are supported by a few overall design objectives that we follow: based on the *earth mover's distance*, should be *scale invariant*, the model contains *few parameters*. In the following we explore in detail the properties above.

- *Earth mover's distance (EMD)*: the earth mover's distance is a well-established metric on histograms representing discretized distributions of numerical quantities. Formally, EMD is formulated as a linear programming problem solving a bipartite network flow problem, as for instance the transportation problem. Consider I to be a set of suppliers and J a set of consumers, where the amount of offer of the suppliers is equal or greater than the amount of demand of the consumers, and $c_{i,j}$ is the cost to move supplies from supplier $i \in I$ to consumer $j \in J$. The solution to the transportation problem is given by the set of flows $f_{i,j}$ that minimize the overall cost of transportation:

$$\sum_{i \in I} \sum_{j \in J} c_{i,j} \, f_{i,j} \tag{3.1}$$

subject to the following constraints:

$$f_{i,j} \geq 0, \quad i \in I, j \in J \tag{3.2}$$

$$\sum_{i \in I} f_{i,j} = y_j, \quad j \in J \tag{3.3}$$

$$\sum_{j \in J} f_{i,j} \leq x_i, \quad i \in I \tag{3.4}$$

where x_i represents the total offer capacity of supplier i, and y_j is the total demand capacity of consumer j. The positivity constraint 3.2 enforces the flows to go from the suppliers to consumers only, whereas constraint 3.3 forces a consumer to take only the amount of supplies needed. Constraint 3.4 states that a supplier can only ship for the amount that is in its stock. The final EMD distance is the optimal transport cost normalized by the

total mass capacity of the consumers:

$$EMD(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i\in I}\sum_{j\in J} c_{i,j}\, f_{i,j}}{\sum_{j\in J} y_j} \tag{3.5}$$

In our case, the suppliers and the consumers are the bins of the histograms, having a transportation cost from i to j equivalent to the distance between the two cells of the histograms indexed by the two indices. Other metrics on discrete probability distributions (e.g. χ^2, Pearson or Bhattacharyya distances) or generic distances on vectors (e.g. Euclidean or cosine distance) would not take the similarity of values in nearby bins into account. EMD has proved extremely effective in tasks like image retrieval [36] requiring distances between quantized distributions. Therefore EMD is therefore the canonical choice for measuring distances between node histograms.

- *Scale invariance*: given two NHTs H_1, H_2: if H_1', H_2' are obtained by multiplying all entries in all histograms of H_1, H_2 by a constant c, then we want to obtain $d(H_1, H_2) = d(H_1', H_2')$. In the context of count data, this seems more appropriate than location invariance (defined as invariance under *addition* of a common constant).

- *Few parameters*: the NHT metric should depend only on a small number of tunable parameters and work well under a simple default setting of these parameters. It is the intention that different behaviors of the final metric can be implemented by modifying the TET structure and the $\boldsymbol{\beta}$ weights of the logistic evaluation function. Adding further tunable parameters to the definition of the NHT metric would to some extent only duplicate capabilities we already have through the construction of customized NHTs.

The construction of the NHT metric has two parts: defining a metric between node histograms of a common dimension, and combining these individual metrics into a metric on NHTs.

3.4.1 Node Histogram Metric

The first part is the more important one, as it is here where we have to incorporate the EMD. EMD is most naturally defined on histograms that have an equal

total number of counts (usually normalized to a probability distribution over the histogram bins). For histograms with unequal total mass, the early definition given in [95] does not lead to a proper metric. A modified definition of EMD for distributions with unequal total mass has been proposed in [88] (similarly also in [77]). The modification essentially consists of adding to the histogram with lower total count a virtual bin that has a constant distance to all other bins, and is assigned the difference of total counts in the two histograms. Then standard EMD is applied to the two now equal-sized histograms. If the constant distance of the virtual bin is at least 1/2 of the maximal distance between any of the original bins, then the result is again a proper metric [88]. This existing approach for dealing with histograms with unequal counts is not very well suited for our purpose, since it does not lead to a scale-invariant measure, and in the case of histograms with widely different total counts, the differences in total counts dominate the computed distance, which then becomes less sensitive to the differences in the distributions over the histogram bins.

We therefore introduce a different approach for dealing with histograms of unequal mass that leads to a proper scale-invariant metric, and that allows us to better calibrate the contributions to the overall distance of differences in total counts, and differences in the distributional patterns.

In the following, h denotes the individual node histogram, and H the entire NHTs. For a histogram h we use $c(h)$ for representing the sum of all cell counts in h. Let h_1, h_2 be two histograms of equal dimensions (i.e., h_1, h_2 represents the histograms for the same TET-feature node, threrefore having the same dimensionality and number of bins, but coming from two different TET-values). Then the *relative count distance* is defined as:

Definition 11 *Given two histograms h_1, h_2 of same dimensionality and same number of bins:*

- *if* $c(h_1) = c(h_2) = 0$, $d_{\text{r-count}}(h_1, h_2) = 0$
- *if* $c(h_1) = 1$ *and* $c(h_2) = 0$
  or $c(h_1) = 0$ *and* $c(h_2) = 1$, $d_{\text{r-count}}(h_1, h_2) = 1$
- *else*

$$d_{\text{r-count}}(h_1, h_2) := 1 - \frac{\min(c(h_1), c(h_2))}{\sqrt{c(h_1) \cdot c(h_2)}} \tag{3.6}$$

Proposition 1 $d_{\text{r-count}}$ *is a scale-invariant pseudo-metric with values in* $[0,1]$.

Proof. The minimum of two counts is a positive semi-definite kernel, called histogram intersection kernel [6]. The normalization is called cosine normalization, and the result is also a kernel [102]. Let us refer to this kernel as

$$k(h_1,h_2) = \frac{min(c(h_1),c(h_2))}{\sqrt{c(h_1)\cdot c(h_2)}}.$$

A kernel induces a pseudo-metric

$$d(h_1,h_2) = \sqrt{k(h_1,h_1)+k(h_2,h_2)-2k(h_1,h_2)}.$$

For the normalized histogram intersection kernel we have that $0 \leq k(h_1,h_2) \leq 1$ and $k(h_1,h_1) = k(h_2,h_2) = 1$, thus $d(h_1,h_2) = \sqrt{2-2k(h_1,h_2)}$. The count distance is obtained as $d_{r\text{-}count}(h_1,h_2) = \frac{1}{2}d(h_1,h_2)^2$, a simplified version of the distance which preserves its properties. The scale invariance property states that the distance between the histograms should remain the same under scalar multiplication. We indicate the newly obtained histograms as jh_1 and jh_2 when thery are multiplied by the constant j. We can therefore rewrite the kernel as

$$k(jh_1,jh_2) = \frac{min(c(jh_1),c(jh_2))}{\sqrt{c(jh_1)\cdot c(jh_2)}}.$$

The count operation on the scaled version is equivalent to multiplying j to the count of the unscaled histograms, and the minimum of the scaled histograms is equivalent to multiplying j to the minimum of the unscaled ones. Therefore the equation becomes

$$k(jh_1,jh_2) = \frac{j\cdot min(c(h_1),c(h_2))}{\sqrt{j^2\cdot c(h_1)\cdot c(h_2)}}.$$

The constants at the enumerator and denominator cancels out, therefore obtaining again the unscaled form of the kernel. As for a proper metric, non-negativity and symmetry are preserved, its demonstrations is trivial. For triangle inequality $d(h_1,h_3) \leq d(h_1,h_2)+d(h_2,h_3)$ implies that $\alpha d(h_1,h_3)^2 \leq \alpha(d(h_1,h_2)+d(h_2,h_3))^2 \leq \alpha d(h_1,h_2)^2 + \alpha d(h_2,h_3)^2$ for any $\alpha > 0$. Finally, $d_{r\text{-}count}$ is a pseudo-metric because any two distinct histograms having same counts have zero distance. One of the properties a metric should satisfy is

the identity of indiscernibles, i.e., $d(h_1,h_2) = 0 iff h_1 = h_2$. A pseudo-metric differs from a proper metric by relaxing this assumption, therefore allowing $d(h_1,h_2) = 0$ for $h_1 \neq h_2$.

Now let $\bar{h} = h/c(h)$ be the probability distribution on histogram bins obtained by normalizing h. The earth mover's distance between these normalized histograms is defined in terms of an underlying ground distance between histogram bins [95]. We take the Manhattan metric as the ground distance, because this is a very commonly used distance on histogram bins, and because it supports a computationally efficient approximation to the EMD that we will introduce in Section 3.4.3 below. Note however that our exact formulation is generic and can be used with any ground distance, provided an appropriate normalization is introduced. To ensure a common scale for all EMDs, regardless of the dimensionality (D) and granularity (N) of the histograms involved, we divide the raw Manhattan distance by $D(N-1)$, so that all distances between histogram bins fall into the interval $[0,1]$. The EMD distance $d_{emd}(\bar{h}_1,\bar{h}_2)$ between two normalized histograms then is defined (and computed) as the solution of a linear program in M^2 variables, where M is the number of bins in each histogram.

Combining the count and the EMD distances via a simple mixture construction, we define:

$$d_{c\text{-}emd}(h_1,h_2) := \frac{1}{2}(d_{r\text{-}count}(h_1,h_2) + d_{emd}(\bar{h}_1,\bar{h}_2)) \tag{3.7}$$

This definition gives equal weight to the $d_{r\text{-}count}$ and d_{emd} components. In the experimental section we will provide results on the application of $d_{c\text{-}emd}$ with equal weights for the components as well as two versions in which only one of the two components is activated.

Proposition 2 *$d_{c\text{-}emd}$ is a scale-invariant metric with values in* $[0,1]$.

The fact that d_{emd} on normalized histograms is a proper metric, and the observation that if $h_1 \neq h_2$ then $c(h_1) \neq c(h_2)$, in which case $d_{r\text{-}count}(h_1,h_2) > 0$, or $\bar{h}_1 \neq \bar{h}_2$, in which case $d_{emd}(\bar{h}_1,\bar{h}_2) > 0$. Thus, $d_{c\text{-}emd}$ is a metric, even though its components only are pseudo-metrics.

3.4.2 Histogram Tree Metric

The last part involves the combination of the node histogram metrics to obtain a metric for histogram trees. Consider two NHTs H_1, H_2 obtained as approximations of logistic evaluation trees $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}_1)))$ and $L^{\boldsymbol{\beta}}(V(T(\boldsymbol{v}_2)))$ for entities $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Since both trees are derived from the same underlying TET T with nodes $(v_0, \ldots, v_k)$, they have identical structure, and consist of node histograms $(h_{1,0}, \ldots, h_{1,k})$ and $(h_{2,0}, \ldots, h_{2,k})$. The combination of the node metrics should be carefully calibrated in order to avoid unwanted behaviours. For instance, performing the plain summation of the node metrics is undesirable, in that branches with many children would dominate the overall distance value. In order to prevent this effect, we scale $d_{c\text{-}emd}(h_{1,i}, h_{2,i})$ by a factor $1/s_i$, where s_i is the number of siblings of v_i in T. This leads to the following definition of a metric between NHTs, which we also denote as $d_{c\text{-}emd}$:

$$d_{c\text{-}emd}(H_1, H_2) := \sum_{i=1}^{k} \frac{1}{s_i} d_{c\text{-}emd}(h_{1,i}, h_{2,i}). \tag{3.8}$$

Proposition 3 *$d_{c\text{-}emd}$ is a scale-invariant metric on node histogram trees.*

3.4.3 Marginal EMD

The computation of $d_{c\text{-}emd}(H_1, H_2)$ with $H_i = (h_{i,1}, \ldots, h_{i,k})$ requires the computation of k EMDs. For a pair of node histograms, the computation consists of a linear optimization problem in N^2 variables, with N the number of bins in the histograms. Assuming a fixed granularity of N bins in each dimension, this computation becomes exponential in the dimensionality of node histograms. In contrast, EMD for 1-dimensional histograms assuming the Manhattan distance as underlying distance can be computed without the use of linear optimization simply by summing over the absolute difference of the cumulative distribution function [25]: for normalized 1-dimensional histogram $\bar{h}_1, \bar{h}_2$ with N cells and cell values $\bar{h}_i(1), \ldots, \bar{h}_i(N)$ $(i = 1, 2)$, define the cumulative cell counts $f_i(k) := \sum_{j=1}^{k} h_i(j)$ $(k = 1, \ldots, N; i = 1, 2)$. Then

$$d_{emd}(\bar{h}_1, \bar{h}_2) = \sum_{k=1}^{N} |f_1(k) - f_2(k)|. \tag{3.9}$$

We can approximate $d_{emd}(\bar{h}_1, \bar{h}_2)$ by considering the 1-dimensional marginals of the $\bar{h}_i$ as follows: for a D-dimensional normalized histogram $\bar{h}$ with N bins in each dimension, and $1 \leq k \leq D$ let $\bar{h}^{\downarrow k}$ denote the marginal of $\bar{h}$ in the kth dimension, i.e., $\bar{h}^{\downarrow k}$ is the 1-dimensional histogram whose count in the jth bin is the sum of all counts in $\bar{h}$ over bins with index j in the kth dimension. For a two-dimensional histogram, for instance, this corresponds to computing row and column sums.

We then define the *marginal EMD distance* between $\bar{h}_1, \bar{h}_2$ as

$$d_{memd}(\bar{h}_1, \bar{h}_2) := \sum_{k=1}^{D} d_{emd}(\bar{h}_1^{\downarrow k}, \bar{h}_2^{\downarrow k}). \tag{3.10}$$

Proposition 4 *d_{memd} is a pseudo-metric with $d_{memd} \leq d_{emd}$.*

It should be emphasized that the inequality $d_{memd} \leq d_{emd}$ depends on our use of the Manhattan distance as the underlying metric in the EMD definition. For other metrics on histogram bins, this inequality need not hold.

3.4.4 Baseline Count Distance

In the preceding sections we have introduced a quite sophisticated metric that starts with the underlying counts-of-counts feature represented by the TET value γ, applies the customizable feature transformation through the logistic evaluation function, and then uses a combination of count based and distribution based metrics on the histogram representations of the resulting logistic value paths. However, a comparison with simpler distance functions defined on the counts of histogram values would provide a fair comparison and insights on our more complex metric. Therefore we introduce here a baseline distance function on node histograms and a baseline function on node histrogram trees, constructed in analogy to the functions presented earlier:

$$d_{b\text{-}count}(h_1, h_2) := (c(h_1) - c(h_2))^2 \tag{3.11}$$

$$d_{b\text{-}count}(H_1, H_2) := \sqrt{\sum_{i=1}^{k} d_{b\text{-}count}(h_{1,i}, h_{2,i})} \tag{3.12}$$

Note that $d_{b\text{-}count}$ is independent of the distribution of counts over the bins in the histogram, and thereby does not depend on the logistic evaluation function

that induces this distribution. Moreover, $d_{b\text{-}count}$ disregards the nested counts-of-counts structure represented in the underlying TET value γ, and only considers "flat" counts.

3.5 Metric Tree Retrieval

Part of the experiments in Section 3.6 focus on efficient nearest neighbor retrieval according to the NHT metric. Computing exact nearest neighbor can become expensive with a large number of data points, therefore we mitigate this problem using an approximate retrieval strategy. Methods that perform approximate nearest neighbor search are conceived for Euclidean spaces [121], relying on tree-decomposition [31] or locality-sensitive-hashing [121, 122] techniques. We use a simple *metric tree* (MT) structure based on generalized hyperplane decomposition [116] for performing the nearest neighbors search. In many information retrieval contexts it is not strict to retrieve all the nearest neighbors, instead, the prediction is based on the k best matches for a given query. The procedure to construct the MT structure is shown in Algorithm 2 The MT is built from a dataset of node histogram trees, by recursively splitting

Algorithm 2 Metric Tree search.

```
1: procedure MTSEARCH (MN,H,k)
2:     if ISLEAF(MN) then
3:         sorted = SORT(MN.bucket, H)
4:         return sorted[1:k]
5:     if DIST(H,MN.z1) ≤ DIST(H,MN.z2) then
6:         return MTSEARCH(MN.left,H,k)
7:     else
8:         return MTSEARCH(MN.right,H,k)
```

data until a stopping condition is met. The parameters of the algorithm are the maximal tree depth (d_{max}) and the maximal bucket size (n_{max}), the current depth (initialized at $d = 1$) and the data to be stored ($data$), one histogram tree for each entity. The MT is composed of internal nodes and leaf nodes. The internal nodes store a single value (in this case, a histogram tree) and has two branches, whereas the leaf nodes contain a set, or bucket, of entities. The MT construction proceeds by splitting data and recursively calling the procedure over each of the subsets, until a stopping condition is met. If the maximal tree

depth is reached, or the current set to be splitted is not larger than the maximal bucket size, a leaf node is returned. If the stopping condition is not met, two entities $z1$ and $z2$ are chosen at random from the set of data (making sure they have a non-zero distance), and data are splitted according to their distances to these entities. Data that are closer to $z1$ go to the left branch, the others go to the right one, and the procedure recurses over each of the branches in turn. Once the MT has been built, the fastest solution for approximate k-nearest-neighbor retrieval for a query instance H amounts to traversing the tree, following at each node the branch whose corresponding entity is closer to the query one, until a leaf node is found. The entities in the bucket contained in the leaf node are then sorted according to their distance to the query entity, and the k nearest neighbors are returned. case [76, 83].

Both algorithms have as additional implicit parameter the distance function over NHTs, which can be the exact EMD-based NHT metric or its approximate version based on marginal EMD. Notice that for large databases, explicitly storing the NHT representation of each entity in the leaf buckets can be infeasible. In this case buckets only containt entity identifiers, and the corresponding NHTs are computed on-the-fly when scanning the bucket for the nearest neighbors. Standard caching solutions can be implemented to speed up this step.

3.6 Experiments

We performed a number of experiments to investigate the usefulness of our metrics, and the quality and efficiency of nearest neighbor retrieval when facilitated by the MEMD approximation, and the MT data structure.

3.6.1 Data and Experimental Setup

Datasets

Our first application domain is bibliometrics. We employed the AMiner dataset[1], which consists of a citation network comprising a total of 1,712,433 authors, 2,092,356 papers and 8,024,869 citations. From this large dataset, we extracted the 103,658 authors with h-index > 2, where the h-index (or Hirsch index) of an author is defined as the largest number h of papers having received at least h citations each. In our experiments we are retrieving nearest neighbors for certain *test* entities from a given *training* dataset. For the AMiner dataset, we split our whole set of 103,658 authors into 2/3 (69,106) for training, and 1/3 (34,552) for testing. For all the experiments, we used $N = 5$ as the number of histogram bins. For the MT, we used $d_{max} = 12$ as the maximum tree depth, and $n_{max} = 30$ as the maximal bucket size. The number of histogram bins is an arbitrary choice of the authors, whereas the values of the hyperparameters of the MT retrieval are set to balance the time needed for the construction of the tree and the comparison between the query and the node histogram trees in the leaves of the MT. We performed several tests in order to balance the time needed to construct the tree (d_{max}) and later the time to perform the pairwise comparisons between an object and all the NHTs grouped into a leaf (n_{max}). The values reported above refer to what we found was the best combination of resources allocation (time and computational) and the final performance of the KNN retrieval. We used the publicly available code from [60] for the computation of TET values for data stored in a MySQL database.

The second application domain we considered is the Internet Movie Database (IMDb).[2] We collected the version dated February 17, 2017, from which we extracted tables regarding movies, genres, actors, and business. We built a dataset

[1] https://aminer.org/aminernetwork
[2] http://www.imdb.com/

of 246,285 movies (those having at least one genre attribute), and considered 9,601 actors (those appearing in more than 20 of those movies). Based on an actor's billing position in the movie's credits we constructed actor-movie relations $\texttt{lead}(a,m)$ if actor a appears in billing position 1 or 2 of movie m, and $\texttt{support}(a,m)$ if a's billing position is greater than 2. Furthermore, we use a generic $\texttt{role}(a,m)$ relation when a appears in m, whether or not billing information is available. As for the business information, we assigned each movie having budget information to one of three categories: $\texttt{large\_budget}(m)$ contains movies with a budget in the top 3% within their decade, `medium_budget` contains the next 20% of most expensive movies, and `small_budget` the remaining movies.

TET parameters

For each experiment we define a TET and a parameter assignment for the logistic evaluation function. We compare three strategies to initialize the weights $\boldsymbol{\beta}$, from manually defining the parameters to learning them from the data.

Default: all "bias" parameters β_0 are set to 0, and all "weight" parameters $\beta_1, \ldots, \beta_m$ are set to 1.

Manual: the weights are set manually in such a way that the evaluation values lay in the linear part of the sigmoid, avoiding all the values to be concentrated in the saturation areas. This enforce the values to be spread across the bins of the nodes' histogram to effectively prove the advantage of using metrics based on the earth mover's distance.

Learned: we learn the parameters by applying a loss function on top of the logistic evaluation function, as introduced in Section 3.2.1. In the case of classification we use the cross-entropy loss, whereas for regression we use the mean squared error loss. We use ADAM [68] as stochastic gradient descent technique, for a maximum of 200 iterations and performing 20 random restarts. The best model is chosen according to the loss on the validation set, having 10% of the points of the training set.

Table 3.1 Overview of methods and notations

	Notation	Definition
TET weight assignment	(def)	default parameter assignment
	(man)	manual parameter assignment
	(l-cla)	parameters learned with cross-entropy loss
	(l-mse)	parameters learned with mean squared error loss
Distance Metrics	C-EMD	combined relative count and EMD distance: equation (3.7)
	C-MEMD	equation (3.7) with d_{emd} replaced by d_{memd}.
	RCOUNT	relative count distance as defined by (3.6)
	MEMD	marginal EMD as defined by (3.10)
	BCOUNT	baseline count distance as defined by (3.3)
	WL-GK	Weisfeiler-Lehman graph kernel
NN retrieval	re-...+MT	use of metric tree data structure for fast retrieval of (approximate) nearest neighbors
Non NN methods	CLA	Classificaton based on logistic evaluation function value
	REGR	Regression based on logistic evaluation function value

Before introducing the experimental results, an explanation on the notation is needed. Table 3.1 summarize the NHT metric, the parameter assignments $\boldsymbol{\beta}$ and the predictors based solely on the logistic evaluation function. The only two metrics not based on TETs are BCOUNT and WL-GK, referring to the baseline count distance defined in Equation 3.3 and the Weisfleiler Lehman Graph Kernel respectively. TET based metrics are expressed in conjunction with the initialization method for $\boldsymbol{\beta}$ employed and if the approximate retrieval is applied, e.g., C-MEMD+MT (l-cla) denotes the use of the combination of the count distance and the marginalized emd, using the Metric Tree to perform the retrieval and with parameters learned on the classification task for the logistic evaluation function.

3.6.2 H-index classification

We consider the binary classification task of predicting whether an author has an h-index greater than 7, using the citation TET of Figure 2.1 (a) with different methods for setting its weights. For this task the manually defined weights

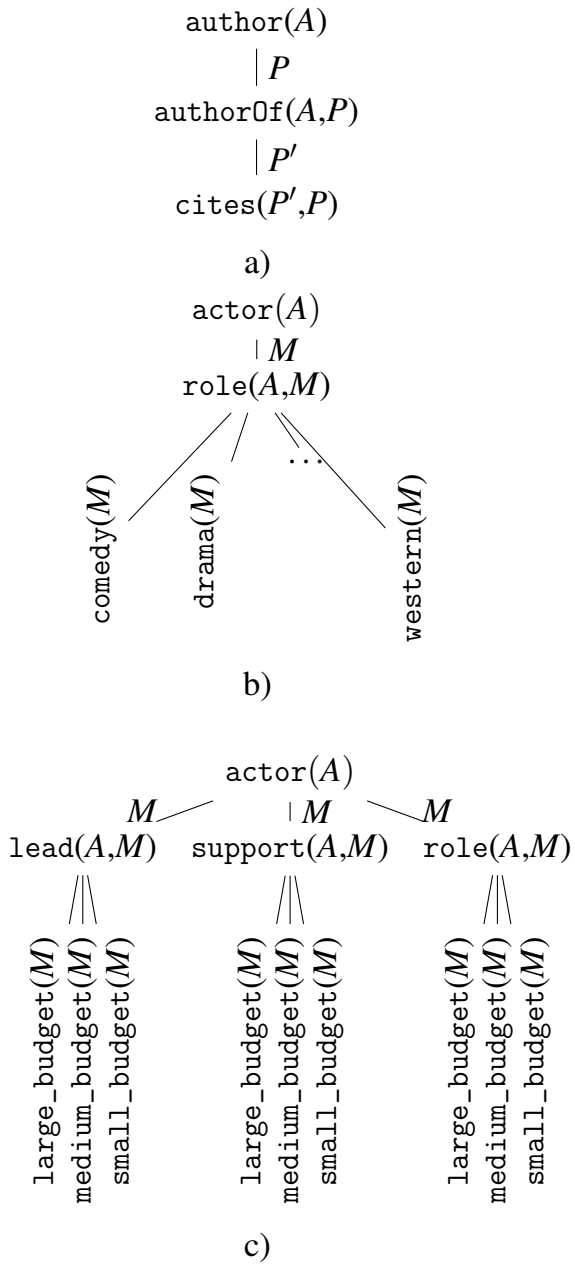


Fig. 3.3 TETs used for AMiner a) and IMDb data b),c).

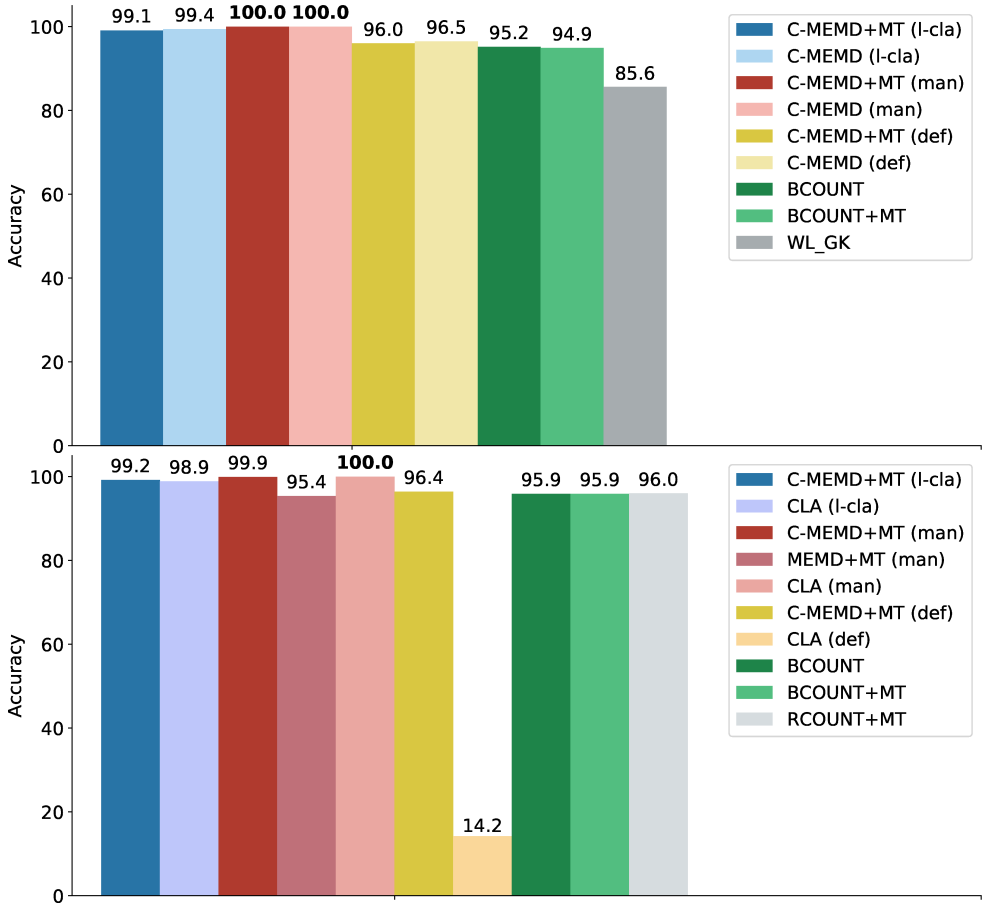


Fig. 3.4 Results on the bibliometrics classification task. We compare the accuracy of the considered methods on the reduced data set (top) and on the full data set (bottom). See Table 3.1 for a definition of the different methods being compared.

are set to approximate a step function going from near 0 at $x = 7$ to near 1 at $x = 8$, thus allowing to achieve perfect classification. We performed for the test authors a k-nearest neighbor prediction based on the $k = 10$ nearest neighbors in the training set, retrieved by C-MEMD+MT. We compare this approach to several alternatives. Some of these alternatives do not scale to our full dataset, and we therefore perform some experiments on a reduced dataset containing 10,000 authors for training, and 1,000 authors for testing. Our first comparison is against C-MEMD to assess the impact of the MT approximation on the prediction accuracy. The second comparison is against nearest-neighbor prediction, when nearest neighbors are determined based on similarity defined by the WL graph kernel (WL-GK). The WL graph kernel expects graphs with a single node attribute and single undirected edges between node pairs. In order to match this format, we represented the relational neighborhood information of an author entity that our citation TET exploits as an undirected graph with nodes representing ground atoms, atom types as node labels and edges connecting ground atoms according to the TET structure. The TET based similarity $d_{c\text{-}memd}$ and the WL-GK based similarity are thus based on the same selection of raw relational data. We used a publicly available graph kernel implementation[3] for this experiment. We set the number of iterations parameter to 1, which we found to give the best results. Our third comparison is with the baseline count metric described in Section 3.4.4. We ran this baseline using both exhaustive nearest neighbor search (BCOUNT) and metric tree approximation (BCOUNT+MT).

Figure 3.4 (top) reports classification accuracy of the methods we tested on the reduced dataset. The first relevant insight is that there is basically no difference between results of the $d_{c\text{-}memd}$ metric using exhaustive or approximate neighbor search. Comparing different weight settings, we see that manually adjusted weights give rise to perfect classification, and that parameter learning finds nearly optimal parameters, substantially better than default ones. Competitors lag clearly behind. The WL-GK is performing the worst, most likely because its aggregation strategy fails to compute the relevant counts-of-counts statistics. The BCOUNT baselines do a reasonable job, but are substantially outperformed by the $d_{c\text{-}memd}$ metric with an appropriate weight setting (manual or learned).

[3]https://github.com/mahito-sugiyama/graph-kernels

Figure 3.4 (bottom) reports the results of a second set of experiments where we use the whole training/test split of the dataset. In this case we compare C-MEMD+MT with different weight settings against three alternatives. The first alternative is given by the same count baselines used in the experiments with the reduced dataset. Results are also very similar, with count baselines achieving the same accuracy as in the reduced dataset. The second alternative consists of simplified versions of the metric in which only the counting component (RCOUNT+MT) or only the memd component (MEMD+MT) is used. In both cases, results are substantially worse than those achieved with the combined metric, confirming the need for considering both aspects of the similarity.

Results achieved with the direct classification model CLA and parameter settings (man) and (l-cla) are very similar to those achieved with nearest-neighbor classification. Both of these parameter settings are optimized for using the logistic evaluation function as a discriminant in this classification task. Under the (def) parameter setting the logistic evaluation always is ≥ 0.5, and therefore CLA (def) classifies all examples as positive, leading to an accuracy of only 14.2. It is noteworthy that even under this parameter setting the similarity based method C-MEMD+MT (def) performs reasonably well, indicating some robustness of the approach with respect to the parameter values.

To summarize, the main insights from the classification experiment are:

- The C-MEMD metric is expressive enough to capture under a suitable parameterization $\boldsymbol{\beta}$ the concept *h-idx* > 7 precisely.
- The classification performance of the metric is quite robust under changes of the parameterization
- Supervised parameter learning under a classification loss function here finds parameters that lead to high classification accuracy, both in the direct classification setting (CLA), and in conjunction with C-MEMD nearest neighbor prediction.
- The counts-of-counts based C-MEMD metric outperforms the flat count metric BCOUNT, where the margin of difference varies with how well the C-MEMD parameters are optimized.

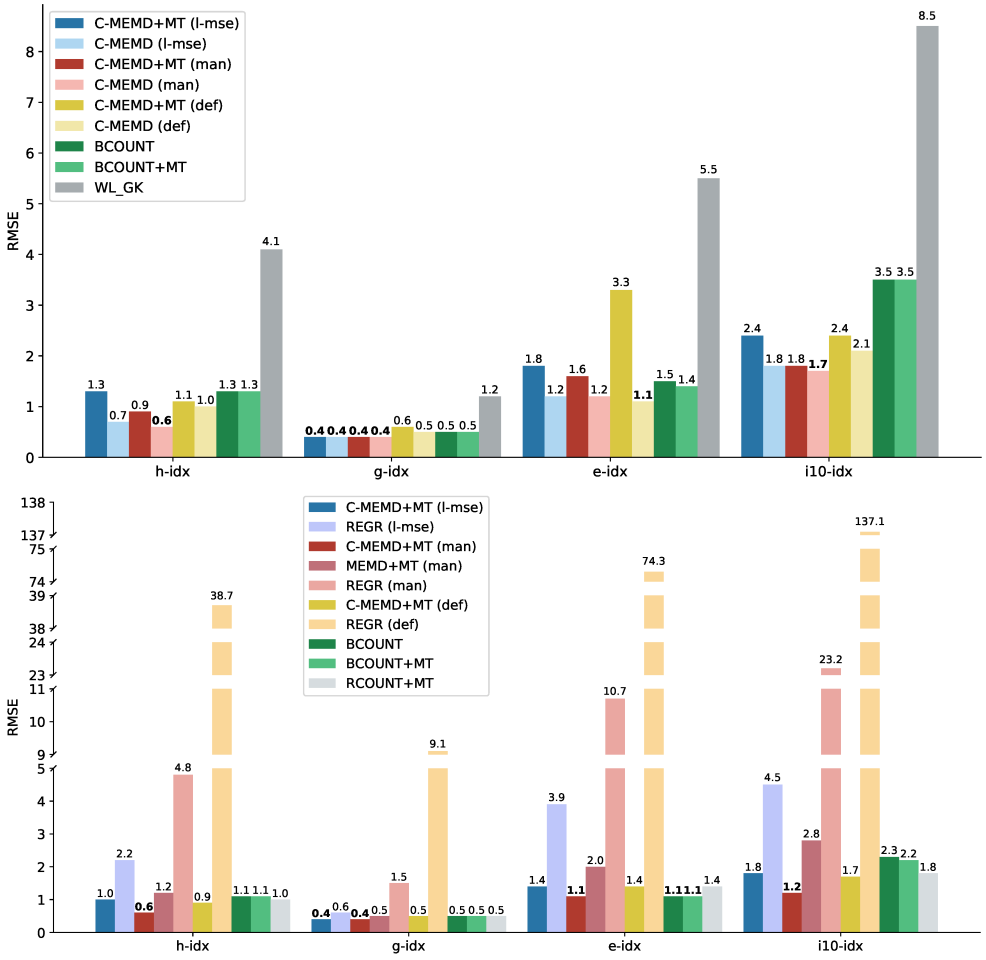


Fig. 3.5 Results on the bibliometrics regression task. We compare the RMSE of the considered competitors on the reduced data set (top) and on the full data set (bottom). See Table 3.1 for a definition of the different methods being compared.

3.6.3 H-index regression

We then addressed a regression task, where the goal is to exactly predict some bibliometric index of an author. We employed h, g, e, and $i10$ indices [55, 43, 130] to investigate the capability of our metric to represent relevant similarity for a broader range of prediction tasks. As before, we use the citation TET of Figure 2.1 (a), with different methods for setting its parameters. Note that in the l-mse weight setting, distinct parameters are learned for each index to be predicted. In the nearest neighbor based approaches, we predict each bibliometric index by the average value of the index in the 10 nearest neighbors. We compare this approach with the same alternatives used for the classification task, both in the reduced and full dataset settings.

Figure 3.5 (top) reports the root mean squared error (RMSE) of the methods we tested on the reduced dataset. A comparison of the performances of C-MEMD and C-MEMD+MT shows that for this more complex task, the MT approximation does produce some performance degradation, albeit differences are rather limited. In terms of weight settings, manual parameters again achieve the best results, and learned weights are very competitive, with almost undistinguishable results in the exact search case. In general, the metric is robust with respect to the choice of parameters, as the default weight setting is also quite competitive. In terms of alternative methods, again the WL-GK is performing the worst, and the BCOUNT baselines do a reasonable job but are outperformed by C-MEMD, especially when combined with exact search.

Figure 3.5 (bottom) reports the root mean squared error (RMSE) of the methods we tested on the full dataset. In this case we compare C-MEMD+MT with different weight settings againts three alternatives. The first alternative is given by the the same count baselines used in the experiments with the reduced dataset. Results are also similar, as C-MEMD+MT with manual parameters outperforms the baselines in all but the e-idx prediction (where they perform the same). The second alternative consists of the same simplified versions of the metric used for the classification case. As in that case, the counting and MEMD components of the metric alone give rise to substantially inferior performance than those achieved with the combined metric. With the direct regression approach REGR we only obtain competitive results with the l-mse weight setting, which substantially outperforms both manual and default parameter settings. This shows the feasibility of weight learning by backpropagation.

However, even under this optimized setting of the weights, the performance of REGR is substantially worse than what we obtain from similarity-based nearest neighbor regression. To summarize, the main insights from the regression experiment are:

- C-MEMD+MT gives competitive results for all three weight setting approaches, demonstrating a certain robustness of C-MEMD based prediction under variations of the weights.
- Similarity-based nearest-neighbor prediction (C-MEMD and BCOUNT) greatly outperforms direct regression (REGR).
- The strong results of C-MEMD+MT under the manual weight setting is evidence for a potential of further improvements that could be obtained by weight optimization using a metric learning (rather than regression) approach.

3.6.4 Retrieval

We consider a task where an objective evaluation criterion can be defined. Named "recasting", this task is designed as follows: the IMDb website[4] lists cases where an actor turned down a role in a major movie, and also states which other actor subsequently filled that role. We can view this as a retrieval problem for a director or casting agent: find for the initially chosen actor (the *query* actor) a similar replacement actor. We denote as *target* actor the actor that ultimately played the role. The IMDb website lists a total of 20 of such query/target pairs of male actors. We omit from our experiments two pairs of actors, reducing the list to 18 pairs. Each target actor is chosen from a set of 9,601 actors. The replacement for a query actor will usually match that query actor both with respect to the type of movies they usually perform in, as well as with respect to commercial aspects as represented by our business TET. We therefore construct a TET incorporates both the actor genre profile and the business information of the film production, as shown in Figure 3.3 b) and c). We construct a TET with a vacuous root set to t, connected to the two sub-TETs describing the profile and business information (b) and c) respectively. Since candidate replacement actors should be active in the same time period as the

[4]https://www.imdb.com/poll/p0HuVFrAcR4/

query actor, we need to consider only movies in a time interval immediately preceding the release date of the query movie. This can be done by adding a temporal feature to the TET, giving a t evaluation if the movie has been produced in a specific temporal window. We use the default parameters for the C-MEMD metric for this experiment. The recasting works as follows: given the query actor, the exhaustive nearest neighbor computation is performed (i.e., without the support of the MT retrieval), thereby defining a distance score for each possible actor in the dataset. This score is used to rank all the actors from most similar to less similar. We compared our C-MEMD metric against the alternative metric based only on the pure count statistics, namely BCOUNT. The rankings obtained by the two metrics correspond to the two columns in Table 3.2. Considering the large number of possible replacement actors, and the fact that the eventual replacement actor (our target) may be the result of many compromises and contingencies faced in the casting process, one should not expect that the target already appears in the top 10, or so, of the nearest neighbor list. In our results, in 11 cases the target actor is in the top 10% of the ranking, whereas in 9 cases he is in the top 5%.

Query actor	Target actor	Movie	C-MEMD	BCOUNT
Al Pacino	Chazz Palminteri	The Usual Suspects	714	1825
Burt Reynolds	Harrison Ford	Star Wars: Episode IV	1545	2263
Daniel Day-Lewis	Viggo Mortensen	Lord of the Rings: the Fellowship of the Ring	1604	9163
Denzel Washington	Brad Pitt	Se7en	32	10
Hugh Jackman	Daniel Craig	Casino Royale	276	8020
Jack Nicholson	Al Pacino	The Godfather	2195	2573
James Caan	Jack Nicholson	One Flew Over the Cuckoo's Nest	348	547
John Travolta	Tom Hanks	Forrest Gump	17	105
Johnny Depp	Matthew Broderick	Ferris Bueller's Day Off	8467	6170
Kevin Costner	Tim Robbins	The Shawshank Redemption	256	129
Leonardo DiCaprio	Mark Wahlberg	Boogie Nights	1635	2077
Leonardo DiCaprio	Christian Bale	American Psycho	235	109
Matt Damon	Sam Worthington	Avatar	3850	3308
Sean Connery	Ian McKellen	The Hobbit: an Unexpected Journey	953	5640
Sylvester Stallone	Brad Pitt	Se7en	247	399
Tom Hanks	Tom Cruise	Jerry Maguire	2	27
Tom Selleck	Harrison Ford	Indiana Jones and the Temple of Doom	162	348
Will Smith	Keanu Reeves	The Matrix	1502	9262

Table 3.2 Results on the recasting experiment. For each movie, we report the ranking of the target actor within the list of the nearest neighbors of the query actor.

3.7 Metric Learning

In this section, we present an extension to the previously defined class of metrics. As described in the previous section, the logistic evaluation function parameters are instantiated following three strategies: default, manual and learned assignments. The experiments in which the parameters were manually assigned obtained the best performances in both classification and regression tasks and performed slightly better than the learned parameters setting. This is not surprising in that the manually assigned parameters were specifically designed to exploit some characteristics of the data to produce meaningful logistic evaluation values and, consequently, histograms. Although manually assigning the parameters is a sensible approach to evaluating the performances, it lacks practical use when the task is more complex and the discriminant rules are harder to define, making it useful for the evaluation but not for the application in real scenarios. In the previous sections, we showed how learning the weights $\boldsymbol{\beta}$ of the logistic evaluation function is performed as a supervised learning task. The output value at the root of the logistic evaluation tree $L^{\boldsymbol{\beta}}$ is compared to the true value via a loss function, and then the parameters are updated via gradient descent. Once the supervised learning task is completed, the best performing model parameters are used for constructing the histograms and comparing two evaluation trees. Although this two-step strategy has been proven effective for calculating the h-index class of an author, an approach in which the parameters are learned via direct application of the metric is desirable. Therefore, here we investigate some possible strategies to perform *metric learning* based on the TET metrics. Metric learning is the task of learning distance functions from a set of data by exploiting supervised or unsupervised learning techniques, becoming a popular task since early two thousands [8]. In the literature, metric learning has been used in many different tasks, e.g., dimensionality reduction, semi-supervised clustering [13, 39, 124], image retrieval [56] and more generally to improve the performances of distance-based classifiers. The vast majority of research is focused on linear metric learning for vectorial data, by learning the covariance matrix of the Mahalanobis distance [126, 37, 124] or a generalized cosine similarity function [90, 27, 7]. Non-linear metric learning has been proposed by kernelizing linear metric learning approaches [104, 26] or performing non-linear projections using, e.g., deep neural networks [30, 97]

or regression trees [64]. Metric learning on structured data has received much less attention due to the challenge given by the combinatorial nature of structured data. A simple option consists of mapping structured objects to feature vectors via kernels on structured data [54] and perform metric learning in feature space. This is, however, highly problematic due to the typically very high-dimensional nature of the feature space and the loss of structural information produced by the feature mapping. Existing works for direct structured metric learning have mostly focused on edit distance learning, where the task is to learn the cost of edits. Initially developed for strings [93], these approaches have been later adapted to deal with trees [18] and graphs [84]. However, these approaches are conceived for measuring distances between entire structures and are extremely expensive when applied to relational data.

3.8 The TETRIC

Our goal is to adapt the TET metrics based on the earth mover's distance to perform metric learning, i.e., learning the parameters of the logistic evaluation while directly measuring the distance between objects. In order to avoid confusion, we call the new adaptated metric introduced here *TETRIC*, the composition of the nouns *TET* and *metRIC*. As for the previous metrics, also the TETRIC defines a distance between two logistic evaluation trees, $L_1^{\boldsymbol{\beta}}, L_2^{\boldsymbol{\beta}}$ obtained from the logistic evaluation function. However, for defining the TETRIC we discard the computation of histograms to balance the model complexity in favor of a simpler strategy. The advantage of using histograms is to fully exploit the earth mover's distance on multidimensional representations. However, relying on the less expressive representation obtained with the marginalization of the histograms, allows for a faster computation while obtaining remarkably good results, as shown in the classification and regression experiments of Section 3.6. Moreover, computing the earth mover's distance involves solving an optimization problem for which computing the derivative would become extremely costly. Another modification we apply in the TETRIC involves the construction of the $\mathscr{M}(v)$, i.e. the multiset of logistic value paths. Computing the value path for a node v_j different from the root $\mathscr{M}(v_j)$ means that all the values from the root to the observed node are replicated during the top down construction procedure, with the cardinality of values in the path correspond-

ing to the dimensions of the histogram, and the actual values determining the position of the cell in the histogram. The marginalization eliminates this positioning, flattening the histograms to a series of bins in a single dimension. Therefore, here we can discard the repetition of the value path in favor of simpler multisets of logistic values, i.e. we do not consider the path of values from the root to the node. To distinguish the multisets of logistic value paths ($\mathscr{M}(v)$) from the multisets of logistic values we use the new notation $M(v)$. We construct the TETRIC using as base components a variation of the pseudo metrics on multisets d_{count} and d_{emd}, defined over pairs $M_1(v), M_2(v)$. The first is just based on the cardinalities of the two multisets:

$$d_{count}(M_1(v), M_2(v)) := 1 - \frac{min(|M_1(v)|, |M_2(v)|)}{\sqrt{|M_1(v)| \cdot |M_2(v)|}}. \tag{3.13}$$

For the second measure, let P_i be the probability distributions on $[0,1]$ defined by normalizing the multiplicities in M_i to probabilities, and let F_i be their cumulative distribution functions. Then

$$d_{emd}(M_1(v), M_2(v)) := \int_0^1 |F_1(x) - F_2(x)| dx. \tag{3.14}$$

is the earth-mover's distance between P_1 and P_2 (with Euclidean distance as the underlying base metric) [25]. We have that d_{count} and d_{emd} are pseudo-metrics on multi-sets of real numbers, and for any coefficients $\lambda_1, \lambda_2 > 0$ the weighted combination $\lambda_1 d_{count} + \lambda_2 d_{emd}$ is a proper metric. A metric between full value trees now is obtained as a linear combination of the d_{count} and d_{emd} distances for all nodes:

$$\begin{aligned} \text{TETRIC}(L_1, L_2) := \sum_{v \in T} (& w_{count}(v) d_{count}(M_1(v), M_2(v)) \\ & + w_{emd}(v) d_{emd}(M_1(v), M_2(v))), \end{aligned} \tag{3.15}$$

where $w_{count}(v), w_{emd}(v) \geq 0$ are node-specific weights for the two distance measures. The TETRIC, thus, is parameterized by the weights defining the logistic evaluation function, and the metric component weights $w_{count}(v), w_{emd}(v)$. We typically constrain the metric component weights to sum up to one, so that TETRIC is normalized to the range $[0,1]$. For a given weighted TET T and two entities x, y we write $\text{TETRIC}(T, x, y)$ for $\text{TETRIC}(L_x, L_y)$, where L_x, L_y are the

value trees obtained from evaluating T for x, y. This notation can be expanded to TETRIC$(T, \boldsymbol{\beta}, \boldsymbol{w}, x, y)$ to make explicit the dependence on the weights of the logistic evaluation function, denoted by $\boldsymbol{\beta}$, and the metric component weights $\boldsymbol{w}$.

The following proposition states two fundamental properties.

Theorem 2 *For any weighted TET T, TETRIC$(T, \cdot, \cdot)$ is a pseudo-metric on domain entities. If $h : V \to V$ is a multi-relational graph automorphism with $x \mapsto x'$, and $y \mapsto y'$, then $d(x,y) = d(x',y')$ and $d(x,x') = d(y,y') = 0$*

3.9 TETRIC Learning

We aim at learning for a given TET T the weights $\boldsymbol{\beta}$ of the logistic evaluation function, and the metric component weights $\boldsymbol{w}$. We assume that supervision is available in the form of *triplet constraints* [105, 9], i.e. triplets of entities h, f, c for which the target distance d should satisfy

$$d(h,c) < d(h,f). \tag{3.16}$$

Triplet constraints are quite flexible, and many forms of supervision can be translated into such constraints. Below we will generate triplet constraints from class label information. In contexts other than classification, they might be generated from qualitative user feedback expressing judgements of relevance or preference. The overall learning process then consists of two parts: generation of training tiplets, and finding weights that satisfy the constraints. We first turn to the second part.

Given a set C of triplet constraints, we use a standard hinge loss

$$l(\boldsymbol{\beta}, \boldsymbol{w}) = \sum_{(h,c,f) \in C} \max(\delta - (\text{TETRIC}(T, \boldsymbol{\beta}, \boldsymbol{w}, h, f) - \text{TETRIC}(T, \boldsymbol{\beta}, \boldsymbol{w}, h, c)), 0), \tag{3.17}$$

where δ is a margin hyperparameter. We are here not adding a regularization term, because we are only learning quite low-dimensional parameter vectors, with a low risk of over-fitting. It is our goal to minimize the loss using a standard batch stochastic gradient descent approach. The critical issue for this is

the computation of the gradients with regard to $\boldsymbol{\beta}$ of d_{emd} terms (3.14), where M_1, M_2 now are multisets in the value trees obtained by evaluating some $v \in T$ for h and f (or h and c) under the current parameters $\boldsymbol{\beta}$. Let $M_1 = x_{1,1} \ldots, x_{1,n_1}$, $M_2 = x_{2,1}, \ldots, x_{2,n_1}$, and let $x_1 \leq x_2 \leq \ldots \leq x_{n_1+n_2}$ be the sorted union of M_1 and M_2. The integral in (3.14) can then be written as

$$\sum_{i=2}^{n_1+n_2} |F_1(x_i(\boldsymbol{\beta})) - F_2(x_i(\boldsymbol{\beta}))|(x_i(\boldsymbol{\beta}) - x_{i-1}(\boldsymbol{\beta})), \tag{3.18}$$

where we also make the dependence on $\boldsymbol{\beta}$ explicit. We first note that the cardinalities $|M_i| = n_i$ $(i = 1,2)$ are determined by the structure of the relational neighborhoods of h, f, c, and do not depend on $\boldsymbol{\beta}$. The weights $\boldsymbol{\beta}$ only affect the values $x_{i,j} = x_{i,j}(\boldsymbol{\beta})$, and, hence their ordering in the joint sequence $x_1 \leq \ldots \leq x_{n_1+n_2}$. Now consider an infinetisimal increment vector $\boldsymbol{\varepsilon}$. Then $|F_1(x_i(\boldsymbol{\beta}+\boldsymbol{\varepsilon})) - F_2(x_i(\boldsymbol{\beta}+\boldsymbol{\varepsilon}))|$ can only differ from $|F_1(x_i(\boldsymbol{\beta})) - F_2(x_i(\boldsymbol{\beta}))|$ if

- there exists $x_{1,j_1}(\boldsymbol{\beta}) = x_{2,j_2}(\boldsymbol{\beta}) = x_i$ $(j_1 \leq n_1,\ j_2 \leq n_2)$, and
- x_{i,j_i} is obtained by evaluating the node $v \in T$ for entity tuples $\boldsymbol{x}_i$ $(i = 1,2)$, such that the relational structures on which the evaluation of v depends are not isomorphic for $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

In other words, the difference $|F_1(x_i(\boldsymbol{\beta})) - F_2(x_i(\boldsymbol{\beta}))|$ is locally constant in $\boldsymbol{\beta}$, unless $\boldsymbol{\beta}$ happens to return at $v \in T$ identical values for two non-isomorphic relational sub-structures.

Even though we here do not develop a fully rigorous formulation and derivation of this statement, we conclude that $|F_1(x_i(\boldsymbol{\beta})) - F_2(x_i(\boldsymbol{\beta}))|$ is locally constant almost everywhere in the parameter space of $\boldsymbol{\beta}$, and, thus, can be treated as a constant when computing the gradient of (3.18). Thus,

$$\frac{\partial}{\partial \boldsymbol{\beta}} d_{emd}(M_1(\boldsymbol{\beta}), M_2(\boldsymbol{\beta})) = \sum_{i=2}^{n_1+n_2} \left(\frac{\partial}{\partial \boldsymbol{\beta}} x_i(\boldsymbol{\beta}) - \frac{\partial}{\partial \boldsymbol{\beta}} x_{i-1}(\boldsymbol{\beta})\right), \tag{3.19}$$

where the gradients on the right-hand side just require partial derivatives of the logistic evaluation function. Note, however, that this expression requires that the values $x_{1,\cdot}(\boldsymbol{\beta}), x_{2,\cdot}(\boldsymbol{\beta})$ are sorted according to the current value of $\boldsymbol{\beta}$.

We next turn to the question of generating training triplets from labeled data. The simplest approach is to randomly sample a reference entity h, and then

Algorithm 3 TETRIC learning.

```
1: procedure TETRIC _LEARN (TET T,training data D, validation set V)
2:     for number of restarts do
3:         β, w=GET_NEXT_INITIALIZATION()
4:         while not converged do
5:             C=GENERATE_TRIPLETS(T, D, β, w)
6:             GRADIENT_DESCENT(T, β, w, C)
7:             EVALUATE(T, β, w, V)
       return best restart result
```

sample c randomly from the entities of the same class as h, and f randomly from entities with a different class. However, in preliminary experiments we found that training on data generated in this way leads to a poor correlation between the loss function, and the actual accuracy achieved on the validation set. In previous work [103, 35] it was suggested that many of such random triplet constraints will be too easy, and one should bias the triplet generation towards hard examples. Specifically, 103 suggest to select for a given h:

$$f = \operatorname{argmin} d(h,\cdot),\ c = \operatorname{argmax} d(h,\cdot), \tag{3.20}$$

where argmin and argmax are taken over the instances with a different, respectively the same, class as h. Generating training triplets in this manner implies an objective that all entities of the same class should be closer to each other than to entities of other classes. However, we find it more realistic, and sufficient for achieving good accuracy in nearest neighbor classification, that a class (according to the target metric) may consist of several coherent clusters. Then it is only required that the nearest neighbor of h belongs to the same class of h, and training triplets can be constructed for a random h by

$$f = \operatorname{argmin} d(h,\cdot),\ c = \operatorname{argmin} d(h,\cdot). \tag{3.21}$$

This may again generate triplets that are too easy. In particular, it may be the case that $d(h,f) - d(h,c) \geq \delta$. Then the triplet h,f,c would not contribute to the loss function, and is omitted. If $0 < d(h,f) - d(h,c) < \delta$, on the other hand, the triplet is retained. Both (3.20) and (3.21) make the triplet generation dependent on the current metric d, and are potentially very expensive due to the minimization/maximization operators. To mitigate these complexities, we

use a metric tree construction for fast, approximate nearest neighbor retrieval, and/or compute the argmin only over a random sub-sample of entities.

Algorithm 3 summarizes our approach. Random initial values for $\boldsymbol{\beta}, \boldsymbol{w}$ are sampled using latin hypercube sampling (line 3). The main loop of lines 4-7 then is a batch stochastic gradient descent, where a batch of training triplets is generated according to the approach described above, one gradient descent step is performed, and the updated weights are evaluated by the accuracy of nearest neighbor classification on the validation set.

3.10 Experiments on Metric Learning

3.10.1 Data and Experimental Setup

We explore the TETRIC application in several scenarios, providing a comparison with the TET metric d_{emd} and state-of-the-art techniques for supervised learning on relational data. To effectively apply the learning procedure based on triplet constraints, we focused on binary classification tasks. The construction of the triplets is performed considering the labels of the examples, as described in Section 3.9. The reference element h and the element closer to the reference, i.e., c, belong to the same class, whereas f is an example of the other class. In theory, the triplets could be constructed also in a multi-class setting, as long as some concept of "closeness" among the classes can be defined in order to determine the tuples of close and far examples used in the hinge loss. We test our metric learning task on different datasets with increasing relation complexity, initially evaluating our method on a synthetic one then moving to some real domains such as movies recommendation and restaurant classification.

3.10.2 Synthetic Dataset

We constructed a synthetic dataset with specific characteristics to asses the ability of the TETRIC to learn meaningful metrics under the constraints imposed by the triplet loss function. This dataset comprises entities of a single type, and three binary attributes characterize the nodes. Each entity of the dataset expresses only one of the three attributes, to which we simply refer to as A,B and C. The number of entities per attribute in the dataset is balanced, for a total of 1000 entities, and each entity represents a node in the relational graph. The relational signature contains one predicate defined on couples of entities, i.e., `edges(N,N')`, which evaluates to t (true) if an edge connects two entities, f (false) otherwise, and three single-variable predicates $A(N)$, $B(N)$ and $C(N)$ which evaluate to t if the entity expresses that attribute. Figure 3.6 shows the TET used to encode the relational features in this experimental setting, extracting, for a given node, the distribution of the three attributes among its neighbors. The task is a binary classification problem, therefore each node is attached with a positive or negative label. The label of a node is determined by the following rule:

- positive if $0.5 < |\mathtt{A}(N')|/max(|\mathtt{B}(N')| + |\mathtt{C}(N')|, 0.01 \times |V|) < 1$
- negative otherwise

where $|\mathtt{A}(N')|$, $\mathtt{B}(N')|$ and $|\mathtt{C}(N')|$ are the number of neighbors showing attribute A, B and C respectively, whereas $|V|$ is the total number of nodes in the graph. The dataset is split into 689 training examples and 311 test examples, using

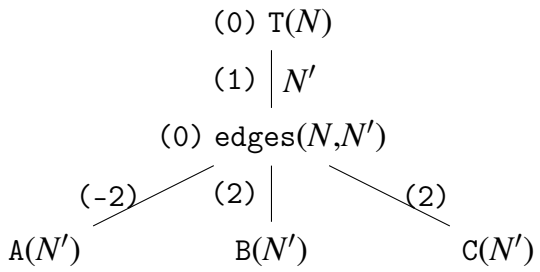


Fig. 3.6 TET for a node of the synthetic dataset. The numbers attached to the nodes and the edges represent the weights of the logistic functions that allow for accurate classification using the d_{c-memd} metric.

20% of the training for validation. For the training, we constructed 1.378 triplets such that each train example is present at least twice in the set of triplets, applying the same strategy to the validation set. The TETRIC is trained using the loss introduced in Section 3.9 for 100 iterations, with a learning rate of 0.01. The expectation of this experiment is that the learned TETRIC should be able to assign lower values when measuring the distance of entities belonging to the same class with respect to entities of the opposite class, in order to achieve perfect prediction when using it as the base metric in the KNN classification algorithm. We use several alternatives of TET-based metrics defined in the previous Section. Our baseline is given by the d_{c-memd} metric presented in Section 3.4.3, for which we know a parametrization of the logistic function (figure 3.6) that allows to reach 100% accuracy with KNN classification. The assignment in the figure is only a possible solution to achieve perfect prediction, therefore we do not expect the TETRIC to learn precisely this assignment, but we know for sure that at least one solution is possible, therefore expecting the TETRIC to be able to learn an assignment that leads to the same results. As we showed in the experimental Section 3.6, we can use both the inner metrics of the TETRIC or disable one of the two. We propose again the same

concept here by using the full TETRIC, i.e., EMD and count (TETRIC$_{cemd}$), the EMD part only (TETRIC$_{emd}$) and the count part only (TETRIC$_{cemd}$). A possible argument against the last two variations is that the main use of the TETRIC is when both the inner metrics are applied, because the flexibility of the TETRIC permits to adjust the contribution given by each one, having as extreme case to totally disable one of the two. This is indeed true, however the previous applications of TET metrics based solely on the EMD or count metric has not been performed under the triplet constraints setting with the hinge loss, therefore justifying this experimental evaluation. Our assumptions were validated, obtaining an accuracy score of 100% with TETRIC$_{cemd}$ and TETRIC$_{emd}$, whereas reaching only 94% with TETRIC$_{count}$. We also perform the classification using only the logistic evaluation function as predictor, applying a cross entropy classification loss, naming this method TET$_{cla}$. This method performs the worst with an accuracy score of just 72%.

3.10.3 Real-world Datasets

As real-world applications, we employed several datasets from different domains. The first benchmark is the AMiner dataset in the same setting as presented in Section 3.6.2 in its reduced version; the second dataset comes from the PAKDD15 competition in which the task is to predict the users' gender by analyzing e-commerce data. In total the dataset contains 30,000 users, divided into 24,000 users for training and 6,000 users for testing. The third benchmark comes from the Yelp challenge, in which we predict if a restaurant sells *chinese* of *mexican* food from the reviews of the users. The restaurants in the datasets can belong only to one of the two classes, therefore no restaurant can sell both chinese and mexican food, and it comprises 3,526 restaurants in the training set and 881 restaurants in the test set. Finally, we predict the gender of users in the movie domain with the Movielens 1M dataset. From the original datasets, all the movies that are not action or dramas genre and that have not been reviewed were removed. Ultimately, the number of users that reviewed the reduced set of movies is 6,040, divided into 4832 users for training and 1208 users for testing. We compare the TETRIC with several alternatives. Being the TETRIC constructed as a variation of the $d_{c\text{-}memd}$ metric, we run the latter with parameters $\boldsymbol{\beta}$ learned by applying the logistic evaluation function with the classification loss. We used three variants of the TETRIC in this ex-

periments; in the first two implementations, we consider the d_{count} or the d_{emd} metric only, represented with the names TETRIC$_{count}$ and TETRIC$_{emd}$ respectively. Then we make use of both the full TETRIC as shown in Equation 3.15 calling it TETRIC$_{cemd}$. In this experiment, we also consider the performance of applying as a discriminant function only the logistic evaluation function as done in the experiments of Section 3.6.2, referring to it as TET$_{cla}$. We compare our methods against some recent techniques for entity prediction in relational graphs. The first competitor is called *Relational Graph Convolutional Newtork* [100] (R-GCN), a new class of graph convolutional networks that use the spatial information of a node's relational neighborhood to construct its embedding. A technique similar to the logistic evaluation function for performing classification and regression tasks is presented by Kazemi et al. in [63], called *Relational Neural Networks* (RelNN), that use hierarchical evaluation of logical groundings to predict a numerical value. While RelNN and R-GCN operate directly on the data structure to obtain a certain response, other approaches try to obtain meaningful latent representations of the graph vertices. An efficient and powerful strategy to obtain such representations is RDF2VEC [94], which extracts random walks from the graph and uses the Skipgram model [81] to produce embeddings of the graph entities, which are then used in a learning model that operates on vectors. In this case, we used a linear SVM as the learning model on top of the entity embeddings. The dataset have all been split in 80% of the examples for training and 20% for testing, following the same strategy used in [63]. We used a different experimental setting for training RDF2VEC, R-GCN, and RelNN with respect to and the TETRICS. Being this methods quite different in their methodology, setting the same learning hyperparamters for all of them would lead to an unclear picture on the actual capabilities of the models. Therefore, for training the TETRICS we let the training algorithm run for a maximum of 100 iterations with a learning rate of 0.05 and performing early stopping on the validation set, which corresponds to the 20% of the training set. The results of the classification tasks on the real world datasets are reported in Table 3.3. On the AMiner dataset, the implementation of RDF2VEC run out of memory, therefore we cannot provide its accuracy score. In general the models based on TETs perform better than the competitors, apart for the gender prediction task in Movielens in which RDF2VEC with linear SVM is superior. Considering the hindex prediction in

MODEL	AMINER	PAKDD	YELP	MOVIELENS
RDF2VEC	NA	86.7	61.5	**81.87**
R-GCN	85.8	87.1	66.4	77.4
RELNN	98.77	88.53	69.27	79.02
TET_{cla}	**99.6**	88.6	71.73	80.31
$TETRIC_{emd}$	99.14	86.63	75.36	76.57
$TETRIC_{count}$	95.97	**89.06**	73.89	73.67
$TETRIC_{cemd}$	98.83	89.03	**76.73**	74.42

Table 3.3 Experiments on metric learning on real datasets. The best performance for each dataset is highlighted in bold.

AMiner, using the logistic evaluation function only we can achieve almost perfect prediction with 99.6% of accuracy. The TETRIC is the best performer in the case of the YELP and PAKDD datasets. The performances of all methods in PAKDD are quite close to each other, although with the TETRIC we can see a slight improvement over the other methods. It is interesting to see how the best performer here is $TETRIC_{count}$ and the second best at only 0.06% distance is $TETRIC_{cemd}$, whereas the $TETRIC_{emd}$ is the overall worse, indicating that in this case the learned weights that balance the two metrics in $TETRIC_{cemd}$ were in fact ignoring the evaluation of the EMD in favor of the plain count distance. Moreover, this can be seen in all the other benchmarks, where the $TETRIC_{cemd}$ places in between the two variants. A significant improvement by using both the EMD and count distance is given in the YELP dataset, in which the combination of both yeld a significant improvement over all the TETRIC variants and the other methods, even the classification model of TET_{cla}.

3.11 Remarks

This chapter illustrates a new class of metrics defined over relational data. The underlying counts-of-counts features constructed from the relational graph allow for the definition of complex hierarchical structures, upon which neural-like functions can be applied on and that can be exploited to construct multi dimensional histograms. We showed how the earth mover's distance is a perfect candidate to measure the dissimilarity between this histograms and how we can combine EMD and other count metrics to effectively compute similarities between tuples of relational entities. Moreover, we showed how we can perform metric learning tasks by modifying the metrics to better adapt to complex domains. The evaluation on both synthetic and real-world datasets proved the effectiveness of the methods.

Chapter 4

Learning Aggregation Functions

4.1 The problem of aggregation

Since the early two thousand, the developments in the machine learning field highlighted how aggregating information has become a relevant part of many learning models, such as fuzzy logic, graph neural networks and, more in general, every technique that operates on sets of data. A large body of literature in recommendation systems uses aggregations to make predictions, e.g., group recommendation, multi-criteria decision making, and score prediction systems. The need to aggregate representations is therefore ubiquitous in deep learning. Some recent examples include max-over-time pooling used in convolutional networks for sequence classification [67], average pooling of neighbors in graph convolutional networks [70], max-pooling in Deep Sets [128], in (generalized) multi-instance learning [114] and in GraphSAGE [53]. In all the above cases (except for LSTM-pooling in GraphSAGE) the aggregation function is predefined, i.e., not tunable, which may be in general a disadvantage [57]. Sum-based aggregation has been advocated based on theoretical findings showing the permutation invariant functions can be sum-decomposed [128, 127]. However, recent discoveries [120] showed that the universal function representation guarantee requires either highly discontinuous (and thus poorly learnable) mappings, or a latent dimension equal to the maximum number of elements in the set. This suggests that the learning of accurate functions on sets of large cardinality is difficult. Inspired by previous work on learning uninorms [80], in this chapter we propose a new parametric family of aggregation functions that we call LAF, for *learning aggregation functions*. A single LAF function can approximate standard aggregators like sum, max, or mean, and it can model more complex functions with non-standard behaviour. Besides, LAF layers with multiple aggregation units can approximate higher-order moments of distributions like variance, skewness, or kurtosis. In contrast, other authors [33] suggest employing a predefined library of elementary aggregators to be combined. Since LAF can represent sums, it can be interpreted as a smooth version of the class of functions that are shown in [128], retaining the universality results in representing set functions. Our empirical findings show that our model is able to generalize over large sets better than other static methods. In particular, we run an extensive experimental analysis showing that LAF layers can learn a wide range of aggregators (including higher-order moments) on sets of scalars with-

out background knowledge on the nature of the aggregation task. Moreover, LAF layers on the top of traditional layers can learn the same wide range of aggregators on sets of high dimensional vectors (MNIST images). Our method outperforms state-of-the-art set learning methods such as DeepSets and PNA on real-world problems involving point clouds and text concept set retrieval. LAF performs comparably to PNA on random graph generation tasks, outperforming several graph neural networks architectures including GAT [119] and GIN [127].

4.2 Learning Aggregation Functions

4.2.1 The Learnable Aggregator

We use $\boldsymbol{x} = \{x_1, \ldots, x_N\}$ to denote finite multisets of real numbers $x_i \in \mathbb{R}$. An aggregation function *agg* is any function that returns for any multiset $\boldsymbol{x}$ of arbitrary cardinality $N \in \mathbb{N}$ a value $agg(\boldsymbol{x}) \in \mathbb{R}$. Note that directly taking $\boldsymbol{x}$ to be a multiset, not a vector, means that there is no need to define properties like exchangeability or permutation equivariance for operations on $\boldsymbol{x}$. An aggregation function *agg* is any function that returns for any multiset $\boldsymbol{x}$ of arbitrary cardinality $N \in \mathbb{N}$ a value $agg(\boldsymbol{x}) \in \mathbb{R}$. A simple approach to model standard aggregation functions like the *mean* or the *max* is to model them in the form of a L_p-norm.

Definition 12 *Given a real number $p \geq 0$, an L_p norm is defined as*

$$L_p(\boldsymbol{x}) := \left(\sum_i x_i^p \right)^{1/p} \tag{4.1}$$

For instance, the maximum value of a set can be computed with the *maximum-norm* in which $p = \infty$. However, we are interested in developing a family of functions that comprehend L_p-norms as a a of possible solutions, but that can represent a larger space of possible functions. A simple modification to the L_p-norm allows for expanding the set of representable functions, and this modification consists in representing the inner exponent and the outer exponent of the function with two separate variables. We therefore build our parametric LAF-aggregator around generalized L_p-norms.

Name	Definition	a	b	c	d	e	f	g	h	α	β	γ	δ	limits
constant	$c \in \mathbb{R}$	0	1	-	-	0	1	-	-	c	0	1	0	
max	$\max_i x_i$	$1/r$	r	-	-	0	1	-	-	1	0	1	0	$r \to \infty$
min	$\min_i x_i$	0	1	$1/r$	r	0	1	-	-	1	-1	1	0	$r \to \infty$
sum	$\sum_i x_i$	1	1	-	-	0	1	-	-	1	0	1	0	
nonzero count	$\lvert\{i : x_i \neq 0\}\rvert$	1	0	-	-	0	1	-	-	1	0	1	0	
mean	$1/N \sum_i x_i$	1	1	-	-	1	0	-	-	1	0	1	0	
kth moment	$1/N \sum_i x_i^k$	1	k	-	-	1	0	-	-	1	0	1	0	
lth power of kth moment	$(1/N \sum_i x_i^k)^l$	l	k	-	-	l	0	-	-	1	0	1	0	
min/max	$\min_i x_i / \max_i x_i$	0	1	$1/r$	r	$1/s$	s	-	-	1	1	1	0	$r, s \to \infty$
max/min	$\max_i x_i / \min_i x_i$	$1/r$	r	-	-	0	1	$1/s$	s	1	0	1	1	$r, s \to \infty$

Table 4.1 Different functions achievable by varying the parameters in the formulation in Eq. 4.3

Definition 13 *A LAF-aggregator for an input set $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ is a function parametrized by two values a, b of the form*

$$L_{a,b}(\boldsymbol{x}) := \left(\sum_i x_i^b \right)^a \qquad (a,b) \geq 0. \tag{4.2}$$

$L_{a,b}$ is invariant under the addition of zeros: $L_{a,b}(\boldsymbol{x}) = L_{a,b}(\boldsymbol{x} \cup \mathbf{0})$ where $\mathbf{0}$ is a multiset of zeros of arbitrary cardinality. In order to also enable aggregations that can represent *conjunctive* behavior such as *min*, we make symmetric use of aggregators of the multisets $\mathbf{1} - \boldsymbol{x} := \{1 - x_i | x_i \in \boldsymbol{x}\}$. For $L_{a,b}(\mathbf{1} - \boldsymbol{x})$ to be a well-behaved, dual version of $L_{a,b}(\boldsymbol{x})$, the values in $\boldsymbol{x}$ need to lie in the range $[0,1]$. We therefore restrict the following definition of our *learnable aggregation function* to sets $\boldsymbol{x}$ whose elements are in $[0,1]$:

Definition 14 *Let $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ a set of input values $x_i \in [0,1], i = 1, \ldots, n$. A Learnable Aggregation Function is defined as*

$$\mathrm{LAF}(\boldsymbol{x}) := \frac{\alpha L_{a,b}(\boldsymbol{x}) + \beta L_{c,d}(\mathbf{1} - \boldsymbol{x})}{\gamma L_{e,f}(\boldsymbol{x}) + \delta L_{g,h}(\mathbf{1} - \boldsymbol{x})} \tag{4.3}$$

where $a, \ldots, h \geq 0$, and $\alpha, \ldots, \delta \in \mathbb{R}$ represent tunable parameters.

Table 4.1 shows how a number of important aggregation functions are special cases of LAF (for values in $[0,1]$). We make repeated use of the fact that $L_{0,1}$ returns the constant 1. For max and min LAF only provides an asymptotic

approximation in the limit of specific function parameters (as indicated in the limits column of Table 4.1). In most cases, the parameterization of LAF for the functions in Table 4.1 will not be unique. Being able to encode the powers of moments implies that e.g. the variance of $\boldsymbol{x}$ can be expressed as the difference $1/N\sum_i x_i^2 - (1/N\sum_i x_i)^2$ of two LAF aggregators. Since LAF includes sum-aggregation, we can adapt the results of [128] and [120] on the theoretical universality of sum-aggregation as follows.

Proposition 5 *Let $\mathscr{X} \subset \mathbb{R}$ be countable, and f a function defined on finite multisets with elements from $\mathscr{X}$. Then there exist functions $\phi : \mathscr{X} \to [0,1]$, $\rho : \mathbb{R} \to \mathbb{R}$, and a parameterization of* LAF*, such that $f(\boldsymbol{x}) = \rho(LAF(\phi\boldsymbol{x}); \alpha, \beta, \gamma, \delta, a, b, c, d)$, where $\phi\boldsymbol{x}$ is the multiset $\{\phi(x) | x \in \boldsymbol{x}\}$.*

Proof. Let $\mathscr{X} = \{x_0, x_1, \ldots\}$. For $i \geq 0$ let r_i be a random number sampled uniformly from the interval $[0,1]$. Define $\phi(x_i) := r_i$. Let $\boldsymbol{x} = \{a_i : x_i | i \in J\}, \boldsymbol{x}' = \{a'_h : x_h | h \in J'\}$ be two finite multisets with elements from $\mathscr{X}$, where J, J' are finite index sets, and a_i, a'_h denote the multiplicity with which elements x_i, x_h appear in $\boldsymbol{x}$, respectively $\boldsymbol{x}'$. Now assume that $\boldsymbol{x} \neq \boldsymbol{x}'$, but

$$\sum_{i \in J} a_i \phi(x_i) = \sum_{h \in J'} a'_h \phi(x_h), \tag{4.4}$$

i.e.,

$$\sum_{j \in J \cup J'} (a_j - a'_j) r_j = 0, \tag{4.5}$$

where now a_j, respectively a'_j is defined as 0 if $j \in J' \setminus J$, respectively $j \in J \setminus J'$. Since $\boldsymbol{x} \neq \boldsymbol{x}'$, the left side of this equation is not identical zero. Without loss of generality, we may actually assume that all coefficients $a_j - a'_j$ are nonzero. The event that the randomly sampled values $\{r_j | j \in J \cup J'\}$ satisfy the linear constraint (4.5) has probability zero. Since the set of pairs of finite multisets over $\mathscr{X}$ is countable, also the probability that there exists any pair $\boldsymbol{x} \neq \boldsymbol{x}'$ for which (4.4) holds is zero. Thus, with probability one, the mapping from multisets $\boldsymbol{x}$ to their sum-aggregation $\sum_{x \in \boldsymbol{x}} \phi(x)$ is injective. In particular, there exists a set of fixed values $r_0, r_1, \ldots$, such that the (deterministic) mapping $x_i \mapsto r_i$ has the desired properties. The existence of the "decoding" function ρ is now guaranteed as in the proofs of [128, 120].

Clearly, due to the randomized construction, the theorem and its proof have limited implications in practice. This however, already is true for previous results along these lines, where at least for the decoding function ρ, not much more than pure existence could be demonstrated. A proof in [120] for a very similar proposition used a mapping from $\mathscr{X}$ into the reals. Our requirement that LAF inputs must be in $[0,1]$ requires a modification of the proof, which for the definition of ϕ relies on a randomized construction.

Proposition 5 shows that we retain the theoretical universality guarantees of [128], while enabling a wider range of solutions based on continuous encoding and decoding functions.

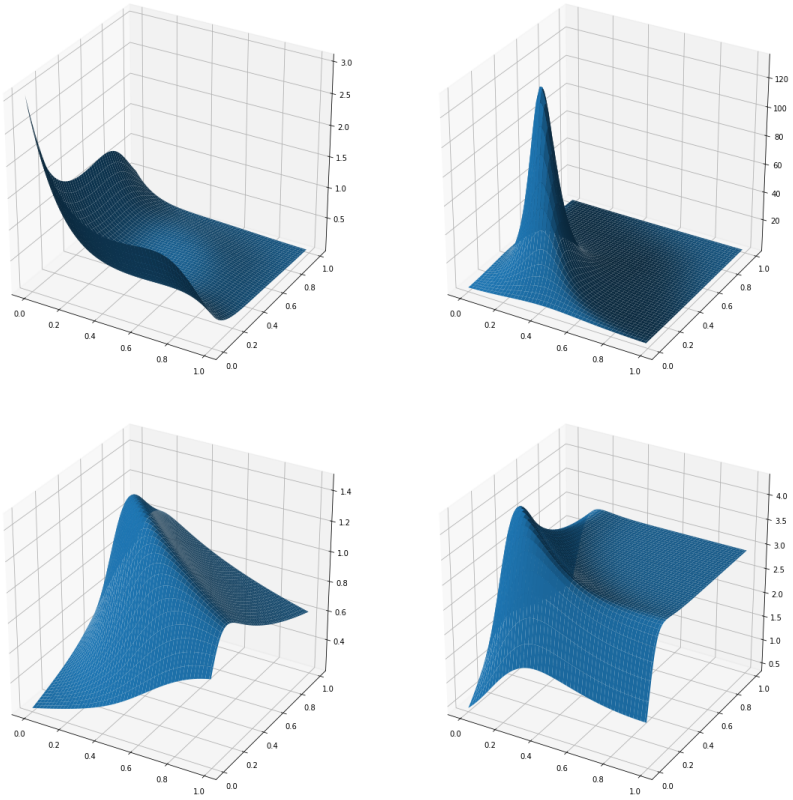


Fig. 4.1 LAF functions with randomly generated parameters

Despite the ability of LAF to incorporate some widely used aggregation functions as special cases, this is not its primary purpose. Indeed the concept of learning is not restricted to enforce LAF to achieve the functions in Table 4.1 but instead, due to its learnable parameters, LAF can represent a continuum of possible aggregators, capable to enable hybrid complex aggregations. An example of several LAF functions that can be obtained with different parametriza-

tions is shown in Figure 4.1. The input space is the unit square $[0,1] \times [0,1]$, i.e., the functions are evaluated over sets of size 2. Parameters $\alpha, \ldots, \gamma$ were randomly sampled in the interval $[0,1]$; parameters b, d, f, h are randomly sampled from the integers $0, \ldots, 5$, and a, c, e, g are obtained as $1/i$ with i a random integer from $0, \ldots, 5$. The figure illustrates the rich repertoire of aggregation functions with different qualitative behaviors already for non-extreme parameter values.

4.2.2 LAF Architecture

Unlike other aggregation functions like *uninorms* [80], LAF is continuous and its derivative can be computed in a closed form, making it amenable to be easily used as a module of any other architecture suitable for learning on sets. Since large portion of machine learning models operate on vectors and not on single scalar values, we describe how to construct a LAF layer that aggregates a set of vector $\boldsymbol{x} = \{x_1, \ldots, x_N\}$ where $x_i \in \mathbb{R}^d$. The aggregation layer can comprise one or several LAF units that can be combined as shown in Figure 4.2. A layer with r LAF units takes as input d-dimensional vectors and produces a vector of size $r \times d$ as output. Each LAF unit performs an *element-wise* aggregation of the vectors in the set such that $L_{k,j} = \text{LAF}(\{x_{i,j}, \ldots, x_{N,j}\}; \alpha_k, \beta_k, \gamma_k, \delta_k, a_k, b_k, c_k, d_k)$ for $k = 1, \ldots, r$ and $j = 1, \ldots, d$. The output vector

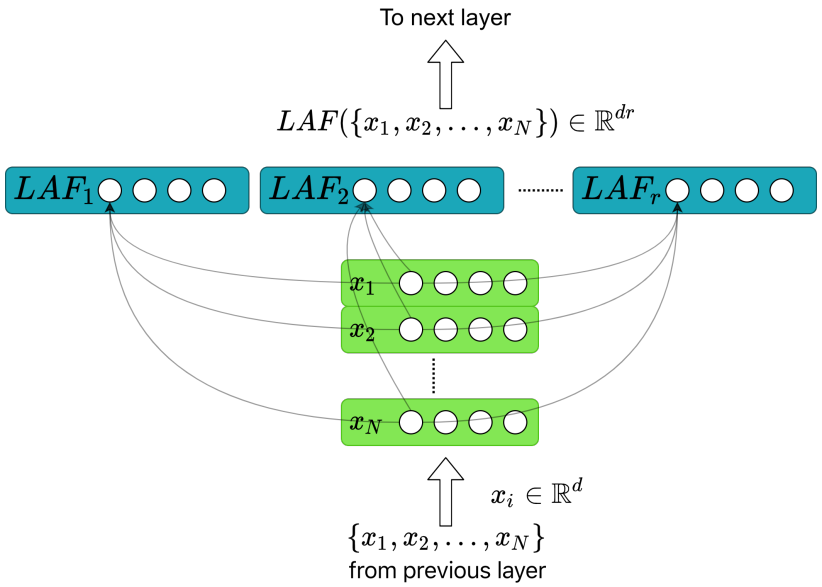


Fig. 4.2 End-to-end LAF architecture.

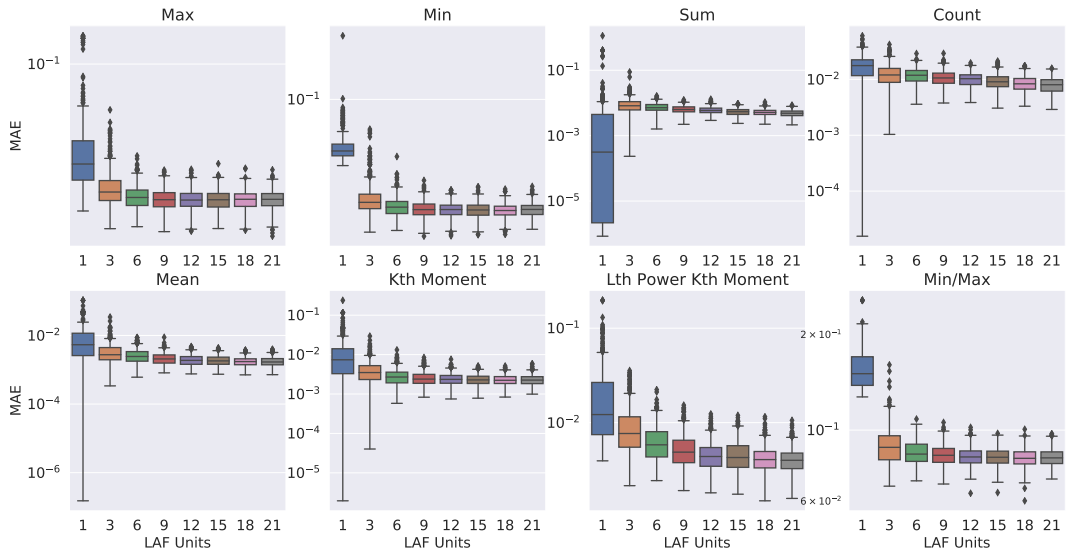


Fig. 4.3 Trend of the MAE obtained with an increasing number of LAF units for most of the functions reported in Table 1. The error distribution is obtained performing 500 runs with different random parameter initializations. A linear layer is stacked on top of the LAF layer with more than 1 unit. The y axis is plot in logaritmic scale.

can be then fed into the next layer. We remark that the values of the input vectors should be in $[0,1]$, therefore when this condition is not satisfied we enforce it by applying a sigmoid before the layer aggregation.

4.2.3 LAF Layer

Learning the functions depicted in Table 4.1 can, in principle, be done by a single LAF unit. However, learning complex aggregation functions might require a larger number of independent units, in that the final aggregation is the result of the combination of simpler aggregations. Moreover, a LAF layer should be able to approximate the behaviour of simpler functions also when multiple units are used. Therefore, we explored the application of multiple LAF units to some of the known functions in Table 4.1.

To perform this analysis, we formulate the task of learning some of the target functions described in Table 4.1. We construct a simple architecture similar to the aggregation layer presented in Section 4.2.1, in which the aggregation is performed using one or more LAF units where, in the case of multiple aggregators, their outputs are combined using a linear layer. We also discard any

non-linear activation function before the aggregation because the input sets are composed of real numbers in the range $[0,1]$, with a maximum of 10 elements for each set. We consider 1,3,6,9,12,15,18 and 21 LAF units in this setting. For each function and each number of units, we performed 500 random restarts. The results are shown in Figure 4.3, where we report the MAE distributions. Let us initially consider the cases when a single unit performs the aggregation. Note first that the functions listed in Table 4.1 can be parametrized in an infinite number of alternative ways. For instance, consider the *sum* function. A possible solution is obtained if $L_{a,b}$ learns the *sum*, $L_{e,f} = 1$ and $\alpha = \gamma$. If instead $L_{a,b} = sum$ and $L_{e,f} = L_{g,h} = 1$, it is sufficient that $\gamma + \delta = \alpha$ to still obtain the sum. This is indeed what we found when inspecting the best performing models among the various restarts. The following are the parametrization, learned for the *sum*, *max* and *mean*, that produced the lowest error with a single unit.

$$sum: \frac{1.751(\sum x^{1.000})^{1.000} + 0.000(\sum(1-x)^{0.000})^{0.560}}{0.909(\sum x^{0.244})^{0.000} + 0.841(\sum(1-x)^{0.364})^{0.000}}$$

$$count: \frac{1.010(\sum x^{0.000})^{0.995} + 0.944(\sum(1-x)^{0.000})^{1.006}}{1.076(\sum x^{0.466})^{0.000} + 0.878(\sum(1-x)^{1.021})^{0.000}}$$

$$mean: \frac{1.515(\sum x^{1.000})^{1.000} + 0.000(\sum(1-x)^{0.618})^{0.000}}{0.000(\sum x^{0.296})^{0.000} + 1.515(\sum(1-x)^{0.000})^{1.000}}$$

On the other hand, the evaluation clearly shows that learning a function with just one LAF unit is not trivial. In some cases, LAF was able almost perfectly to match the target function, but to be reasonably confident to learn a good representation many random restarts are needed, since the variance among different runs is quite large. The error variance reduces when more than one LAF unit is adopted, drastically dropping when six units are used in parallel, still maintaining a reasonable average error. Jointly learning multiple LAF units and combining their outputs can lead to two possible behaviours giving rise to an accurate approximation of the underlying function: in the first case, it is possible that one "lucky" unit learns a parametrization close to the target function, leaving the linear layer after the aggregation to learn to choose that unit or to rescale its output. In the second case, the target function representation is "distributed" among the different units. Here the linear layer is responsible for obtaining the function by combining the LAF aggregation outputs. In the following, we show another example of a learned model, for a setting with three

LAF units. Here the target function is the *count*.

$$unit1: \frac{0.809(\sum x^{0.875})^{0.372} + 0.799(\sum(1-x)^{0.736})^{0.721}}{1.189(\sum x^{0.192})^{0.719} + 1.177(\sum(1-x)^{0.000})^{0.619}}$$

$$unit2: \frac{1.426(\sum x^{0.000})^{1.102} + 1.308(\sum(1-x)^{0.010})^{0.739}}{0.636(\sum x^{0.848})^{0.000} + 0.622(\sum(1-x)^{0.456})^{0.000}}$$

$$unit3: \frac{0.835(\sum x^{0.866})^{0.374} + 0.767(\sum(1-x)^{0.122})^{0.000}}{1.167(\sum x^{0.692})^{0.859} + 1.216(\sum(1-x)^{0.000})^{0.165}}$$

$$linear: 0.0175 + (-0.1265 * unit1) + +(0.5027 * unit2) + (-0.0681 * unit3)$$

In this case, the second unit learns a function that counts twice the elements of the set. The output of this unit is then halved by the linear layer, which gives minimal weights to the other units' outputs.

Although using only one function is sometimes sufficient to approximate the target function significantly, the error variance among different runs is relatively high, indicating that the optimization sometimes fails to converge to a good set of parameters. Multiple units provide more stability while performing better than a single unit aggregation in some cases. We, therefore, construct the LAF architecture for the experimental section by using multiple aggregators, computing the final aggregation as a linear combination of the units' aggregation.

4.3 Experiments

In this section, we present and discuss experimental results showing the LAF framework's potential on both synthetic and real-world datasets. Synthetic experiments are aimed at showing the ability of LAF to learn a wide range of aggregators and its ability to generalize over set sizes (i.e., having test-set sets whose cardinality exceeds the cardinality of the training-set sets), something that alternative architectures based on predefined aggregators fail to achieve. We use DeepSets [128], PNA [33], and LSTM as representatives of these architectures. The LSTM architecture corresponds to a version of DeepSets where an LSTM layer replaces the aggregation function. Experiments on diverse tasks, including point cloud classification, text concept set retrieval, and graph properties prediction, aim to show the potential of the framework on real-world applications.

4.3.1 Scalars Aggregation

This section shows the ability of the LAF framework to learn simple and complex aggregation functions where constituents of the sets are simple numerical values. In this setting we consider sets made of scalar integer values. The training set is constructed as follows: for each set, we initially sample its cardinality K from a uniform distribution taking values in $\{2, M\}$, and then we uniformly sample K integers in $0, \ldots, 9$. For the training set we use $M = 10$. We construct several test sets for different values of M ($M = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$). This implies that models need to generalize to larger set sizes. Contrarily to the training set, each test set is constructed in order to diversify the target labels it contains, so as to avoid degenerate behaviours for large set sizes (e.g., maximum constantly equal to 9). Each synthetic dataset is composed of 100,000 sets for training, 20,000 set for validating and 100,000 for testing. The number of aggregation units is set as follows. The model contains nine LAF (Equation 4.3) units, whose parameters $\{a_k, \ldots, h_k\}$, $k = 1, \ldots, 9$ are initialized according to a uniform sampling in $[0, 1]$ as those parameters must be positive, whereas the coefficients $\{\alpha, \ldots, \delta\}$ are initialized with a Gaussian distribution with zero mean and standard deviation of 0.01 to cover also negative values. The positivity constraint for parameters $\{a, b, ..., h\}$ is enforced by projection during the optimization process. The remaining parameters can take on nega-

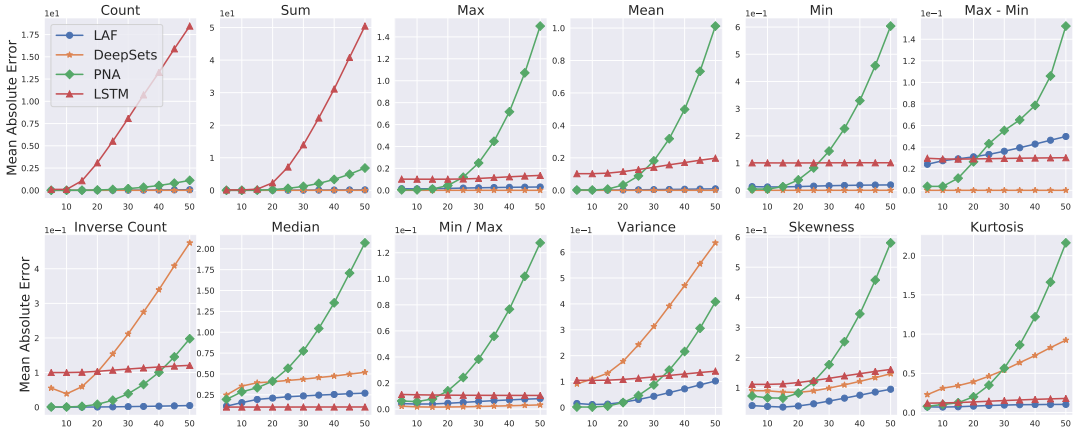


Fig. 4.4 Test performances for the synthetic experiment with integer scalars on increasing test set size. The x axis of the figures represents the maximum test set cardinality, whereas the y axis depicts the MAE error. The dot, star, diamond and triangle symbols denote LAF, DeepSets, PNA, and LSTM respectively.

tive values. DeepSets also uses nine units: three max units, three sum units, and three mean units and PNA uses seven units: mean, max, sum, standard deviation, variance, skewness and kurtosis. Preliminary experiments showed that expanding the set of aggregators for PNA with higher order moments only leads to worse performance. Each set of integers is fed into an embedding layer (followed by a sigmoid) before performing the aggregation function. DeepSets and PNA do need an embedding layer (otherwise they would have no parameters to be tuned). Although LAF does not need an embedding layer, we used it in all models to make the comparison more uniform. The architecture details are reported in the supplementary material. We used mini-batches of 64 sets and trained the models for 100 epochs. We use Adam as parameter optimizer, setting the initial learning rate to $1e^{-3}$ and apply adaptive decay based on the validation loss, and the loss function is the Mean Absolute Error (MAE). The algorithm is trained for 100 epochs. Again, each element in the dataset is a set of scalars $\boldsymbol{x} = \{x_1, \ldots, x_N\}$, $x_i \in \mathbb{R}$ and the learning architecture we designed is the following: Network architecture:

$$\boldsymbol{x} \rightarrow \textsc{Embedding}(10,10) \rightarrow \textsc{Sigmoid}$$
$$\rightarrow \text{LAF}(9) \rightarrow \textsc{Dense}(10 \times 9,\ 1)$$

Figure 4.4 shows the trend of the MAE error for the three methods for increasing test set sizes, for different types of target aggregators. As expected, DeepSets manages to learn the identity function and thus correctly models aggregators like sum, max and mean. Even if LAF needs to adjust its parameters in order to properly aggregate the data, its performance are competitive with those of DeepSets. When moving to more complex aggregators like inverse count, median or moments of different orders, DeepSets fails to learn the latent representation. One the other hand, the performance of LAF is very stable for growing set sizes. While having in principle at its disposal most of the target aggregators (including higher order moment) PNA badly overfits over the cardinality of sets in the training set in all cases (remember that the training set contains sets of cardinality at most 10). The reason why LAF substantially outperforms PNA on large set sizes could be explained in terms of a greater flexibility to adapt to the learnt representation. Indeed, LAF parameters can adjust the *laf* function to be compliant with the latent representation even if the input mapping fails to learn the identity. On the other hand, having a bunch of fixed, hard-coded aggregators, PNA needs to be able to both learn the identity mapping and select the correct aggregator among the candidates. Finally, LSTM exhibits generally poor results when compared to the other methods, particularly in the case of the count and the sum.

4.3.2 MNIST digits

In this section, we modify the previous experimental setting to process MNIST images of digits. The dataset is the same as in the experiment on scalars, but integers are replaced by randomly sampling MNIST images for the same digits. Instances for the training and test sets are drawn from the MNIST training and test sets, respectively. This experiment aims to demonstrate the ability of LAF to learn from more complex representations of the data by plugging it into end-to-end differentiable architectures. The architecture details are the same of the experiment on scalars, the difference is that the elements of the sets in input are vectors, i.e., $\boldsymbol{x} = \{x_1, \dots, x_N\}$, $x_i \in \mathbb{R}^{784}$. Contrarily to the model of the previous section, here we use three dense layers for learning picture

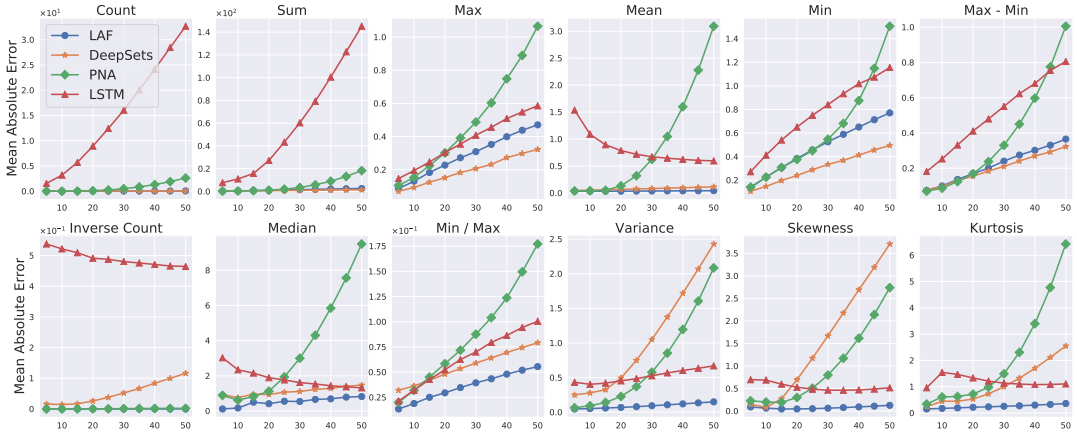


Fig. 4.5 Test performances for the synthetic experiment on MNIST digits on increasing test set size. The x axis of the figures represents the maximum test set cardinality, whereas the y axis depicts the MAE error. The dot, star, diamond and traingle symbols denote LAF, DeepSets, PNA and LSTM respectively.

representations before performing the aggregation:

$$
\begin{aligned}
\boldsymbol{x} &\to \textsc{Dense}(784,300) \to \textsc{Tanh} \\
&\to \textsc{Dense}(300,100) \to \textsc{Tanh} \\
&\to \textsc{Dense}(100,30) \to \textsc{Sigmod} \\
&\to \textsc{LAF}(9) \to \textsc{Dense}(30 \times 9, 1000) \to \textsc{Tanh} \\
&\to \textsc{Dense}(1000,100) \to \textsc{Tanh} \to \textsc{Dense}(100,1)
\end{aligned}
$$

Figure 4.5 shows the comparison of LAF, DeepSets, PNA, and LSTM in this setting. Results are quite similar to those achieved in the scalar setting, indicating that LAF is capable of effectively backpropagating information so as to drive the learning of an appropriate latent representation, while DeepSets, PNA, and LSTM suffer from the same problems seen in aggregating scalars.

Furthermore, Figure 4.6 provides a qualitative evaluation of the predictions of the LAF, DeepSets, and PNA methods on a representative subset of the target aggregators. The images illustrate the correlation between the true labels and the predictions. LAF predictions are distributed over the diagonal line, with no clear bias. On the other hand, DeepSets and PNA perform generally worse than LAF, exhibiting higher variances. In particular, for inverse count and kurtosis, DeepSets and PNA predictions are condensed in a specific area, suggesting an overfitting on the training set.

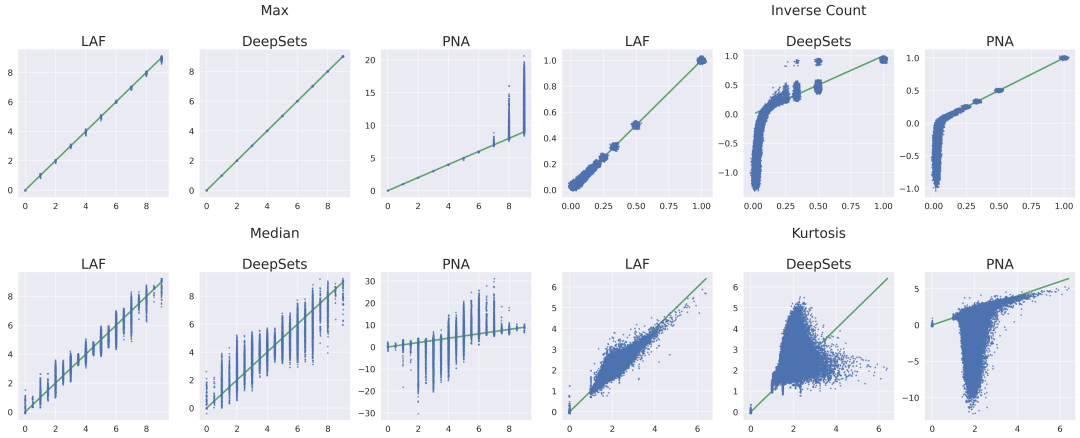


Fig. 4.6 Scatter plots of the MNIST experiment comparing true (x axis) and predicted (y axis) values with 50 as maximum test set size. The target aggregations are *max* (up-left), *inverse count* (up-right), *median* (bottom-left) and *kurtosis* (bottom-right).

4.3.3 Point Cloud

In order to evaluate LAF on real-world dataset, we consider point cloud classification, a prototype task for set-wise prediction. Therefore, we run experimental comparisons on the ModelNet40 [125] dataset, which consists of 9,843 training and 2,468 test point clouds of objects distributed over 40 classes. The dataset is preprocessed following the same procedure described by [128]. We create point clouds of 100 and 1,000 three-dimensional points by adopting the point-cloud library's sampling routine developed by [96] and normalizing each set of points to have zero mean (along each axis) and unit (global) variance. We refer with P100 and P1000 to the two datasets. For all the settings, we consider the same architecture and hyper-parameters of the DeepSets permutation invariant model described by [128]. For LAF, we replace the original aggregation function (max) used in DeepSets with 10 LAF units, while for PNA we use the concatenation of max, min, mean, and standard deviation, as proposed by the authors. For PNA we do not consider any scaler, as the cardinalities of the sets are fixed. For more details about the training setting, refer to [128]. Results in Table 4.2 show that LAF produces an advantage in the lower resolution dataset (i.e. on P100), while it obtains comparable (and slightly more stable) performances in the higher resolution one (i.e. on P1000). These results suggest that having predefined aggregators is not necessarily an optimal

METHOD	P100	P1000
DEEPSETS	82.0±2.0%	87.0±1.0%
PNA	82.9±0.7%	86.4±0.6%
LSTM	78.7±1.1%	82.2±1.7%
LAF	**84.0±0.6%**	**87.0±0.5%**

Table 4.2 Results on the Point Cloud classification task. Accuracies with standard deviations (calculated on 5 runs) for the ModelNet40 dataset.

choice in real world cases, and that the flexibility of LAF in modeling diverse aggregation functions can boost performance and stability.

4.3.4 Set Expansion

Following the experimental setup of DeepSets, we also considered the *Set Expansion* task. In this task the aim is to augment a set of objects of the same class with other similar objects, as explained in [128]. The model learns to predict a score for an object given a query set and decide whether to add the object to the existing set. Specifically, [128] consider the specific application of set expansion to text concept retrieval. The idea is to retrieve words that belong to a particular concept, giving as input set a set of words having the same concept. We employ the same model and hyper-parameters of the original publication, where we replace the sum-decomposition aggregation with LAF units for our methods and the min, max, mean, and standard deviation aggregators for PNA.

We trained our model on sets constructed from a vocabulary of different size, namely *LDA-1K*, *LDA-3K* and *LDA-5K*, having respectively 17k, 38k and 61k words. We used the same training setting as detailed in [128]. Table 4.3 shows the results of LAF, DeepSets and PNA on different evaluation metrics. We report the retrieval metrics recall@K, median rank and mean reciprocal rank. We also report the results on other methods the authors compared to in the original paper. More details on the other methods in the table can be found in the original publication. Briefly, *Random* samples a word uniformly from the vocabulary; *Bayes Set* [50]; *w2v-Near* computes the nearest neighbors in the word2vec [81] space; *NN-max* uses a similar architecture as our DeepSets but uses max pooling to compute the set feature, as opposed to sum pooling; *NN-max-con* uses max pooling on set elements but concatenates this pooled

METHOD	LDA-1k (VOCAB = 17k)					LDA-3k (VOCAB = 38k)					LDA-5k (VOCAB = 61k)				
	RECALL(%)			MRR	MED.	RECALL(%)			MRR	MED.	RECALL(%)			MRR	MED.
	@10	@100	@1K			@10	@100	@1K			@10	@100	@1K		
RANDOM	0.06	0.6	5.9	0.001	8520	0.02	0.2	2.6	0.000	28635	0.01	0.2	1.6	0.000	30600
BAYES SET	1.69	11.9	37.2	0.007	2848	2.01	14.5	36.5	0.008	3234	1.75	12.5	34.5	0.007	3590
W2V NEAR	6.00	**28.1**	54.7	0.021	641	4.80	21.2	43.2	0.016	2054	4.03	16.7	35.2	0.013	6900
NN-MAX	4.78	22.5	53.1	0.023	779	5.30	24.9	54.8	0.025	672	4.72	21.4	47.0	0.022	1320
NN-SUM-CON	4.58	19.8	48.5	0.021	1110	5.81	27.2	60.0	0.027	453	4.87	23.5	53.9	0.022	731
NN-MAX-CON	3.36	16.9	46.6	0.018	1250	5.61	25.7	57.5	0.026	570	4.72	22.0	51.8	0.022	877
DEEPSETS	5.53	24.2	54.3	0.025	696	6.04	28.5	60.7	0.027	426	5.54	26.1	55.5	0.026	616
DEEPSETS*	5.89	26.0	**55.3**	0.026	**619**	7.56	28.5	**64.0**	0.035	349	6.49	27.9	**56.9**	0.030	536
PNA	5.56	24.7	53.2	0.027	753	7.04	27.2	58.7	0.028	502	5.47	23.8	52.4	0.025	807
LSTM	4.29	21.5	52.6	0.022	690	5.56	25.7	58.8	0.026	830	4.87	23.8	55.0	0.022	672
LAF	**6.51**	26.6	54.5	**0.030**	650	**8.14**	**32.3**	62.8	**0.037**	**339**	**6.71**	**28.3**	**56.9**	**0.031**	**523**

Table 4.3 Results on Text Concept Set Retrieval on LDA-1k, LDA-3k, and LDA-5k. Bold values denote the best performance for each metric.

representation with that of query for a final set feature; *NN-sum-con* is similar to NN-max-con but uses sum pooling followed by concatenation with query representation. For the sake of fairness, we have rerun DeepSets using the current implementation from the authors (indicated as DeepSet* in Table 4.3), exhibiting better results than the ones reported in the original paper. We have trained the methods under the same setting of the original implementation, we do not have any satisfying explanation for this phenomenon. Nonetheless, LAF outperforms all other methods in most cases, especially on *LDA-3K* and *LDA-5K*.

4.3.5 Multi-task graph properties

[33] defines a benchmark consisting of 6 classical graph theory tasks on artificially generated graphs from a wide range of popular graph types like Erdos-Renyi, Barabasi-Albert or star-shaped graphs. Three of the tasks are defined for nodes, while the other three for whole graphs. The node tasks are the single-source shortest-path lengths (N1), the eccentricity (N2) and the Laplacian features (N3). The graph tasks are graph connectivity (G1), diameter (G2), and the spectral radius (G3). For more details about the experimental settings please refer to [33].

We compare LAF against PNA by simply replacing the original PNA aggregators and scalers with 100 LAF units (see Equation 4.3). Table 4.4 shows

METHOD	N1	N2	N3	G1	G2	G3
BASELINE	-1.87	-1.50	-1.60	-0.62	-1.30	-1.41
GIN	-2.00	-1.90	-1.60	-1.61	-2.17	-2.66
GCN	-2.16	-1.89	-1.60	-1.69	-2.14	-2.79
GAT	-2.34	-2.09	-1.60	-2.44	-2.40	-2.70
MPNN (MAX)	-2.33	-2.26	-2.37	-1.82	-2.69	-3.52
MPNN (SUM)	-2.36	-2.16	-2.59	-2.54	-2.67	-2.87
PNA (NO SCALERS)	-2.54	-2.42	-2.94	**-2.61**	-2.82	-3.29
PNA	**-2.89**	**-2.89**	**-3.77**	**-2.61**	**-3.04**	-3.57
LAF	-2.13	-2.20	-1.67	-2.35	-2.77	**-3.63**

Table 4.4 Results on the Multi-task graph properties prediction benchmark. Results are expressed in $\log 10$ of mean squared error.

that albeit these datasets were designed to highlight the features of the PNA architecture, that outperforms a wide range of alternative graph neural network approaches LAF produces competitive results, outperforming state-of-the-art GNN approaches like GIN [127], GCN [70] and GAT [119] and even improving over PNA on spectral radius prediction.

4.3.6 SetTransformer with LAF aggregation

In this section we investigate the combination of LAF aggregation with attention mechanisms on sets as proposed in the SetTransformer framework [74]. Briefly, SetTransformer consists of an *encoder* and a *decoder*. The encoder maps a set of input vectors into a set of feature vectors by leveraging attention blocks. The decoder employs a *pooling multihead attention* (PMA) layer, which aggregates the set of feature vectors produced by the encoder. In the following experiment we replace PMA by a LAF layer.

Inspired by one of the tasks described in [74], we propose here to approximate the average of the unique numbers in a set of MNIST images. Solving the task requires to learn a cascade of two processing steps, one that detects unique elements in a set (which can be done by the transformer encoder, as shown in the experiment by [74]) and one that aggregates the results by averaging (which LAF is supposed to do well). The set cardinalities are uniformly sampled from $\{2,3,4,5\}$ and each set label is calculated as the average of the unique digits contained in the set. We trained two SetTransformer models: one

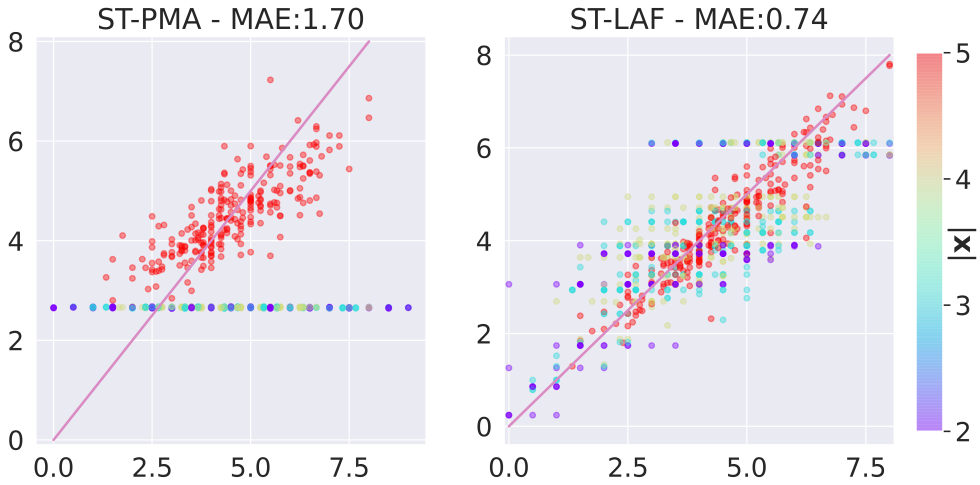


Fig. 4.7 Distribution of the predicted values for ST-PMA and ST-LAF by set cardinalities. On the x-axis the true labels of the sets, on the y-axis the predicted ones. Different colors represent the sets' cardinalities $|\boldsymbol{x}|$.

with PMA (ST-PMA) and the other with LAF (ST-LAF). We used mini-batches of 64 sets and trained the models for 1,000 epochs. We use Adam as parameter optimizer, setting the initial learning rate to $5e^{-4}$. Each element in the dataset is a set of vectors $\boldsymbol{x} = \{x_1, \ldots, x_N\}$, $x_i \in \mathbb{R}^{784}$. This is a detailed description of the architecture of the network:

$$
\begin{aligned}
\boldsymbol{x} &\rightarrow \textsc{Dense}(784,300) \rightarrow \textsc{ReLU} \\
&\rightarrow \textsc{Dense}(300,100) \rightarrow \textsc{ReLU} \\
&\rightarrow \textsc{Dense}(100,30) \rightarrow \textsc{Sigmod} \\
&\rightarrow \textsc{SAB}(64,4) \rightarrow \textsc{SAB}(64,4) \\
&\rightarrow \textsc{PMA}_k(64,4) \text{ OR } \textsc{LAF}(10) \\
&\rightarrow \textsc{Dense}(64 \times k \text{ OR } 9,\ 100) \rightarrow \textsc{ReLU} \\
&\rightarrow \textsc{Dense}(100,1)
\end{aligned}
$$

Please refer to [74] for the SAB and PMA details. Quantitative and qualitative results of the evaluation are shown in Figure 4.7, where we report the MAE for both methods[1]. ST-LAF exhibits a nice improvement over ST-PMA for this particular task. Note that for ST-PMA only 25% of the sets (red points in the scatter plot), corresponding to sets with maximum cardinality, approximates

[1]We run several experiments by changing the number of seeds k of PMA. All of them exhibited the same behaviour. For this experiment we used $k = 1$.

well the average, while for all other cardinalities (the remaining 75% of the sets) ST-PMA predicts a constant value equal to the average label in the training set. ST-LAF instead clearly captures the distribution of the labels, generalizing better with respect to the set sizes. We performed the same experiment substituting the average with the sum, obtaining similar results.

4.4 Remarks

In this chapter we observed how the task of aggregating set information represents a relevant problem in many learning models, and how the choice of the aggregator for a specific problem is ambiguous. Fixed aggregation functions perform well in many cases, but using library-based strategies with many aggregators can potentially produce a loss in the performances of the model, and poorly adapt to diverse set sizes. On the other side, recent discoveries on sum-decomposition as a universal framework for defining set functions demonstrate that it does not necessarily provide a template for practical solutions. Our solution is LAF, a flexible framework for learning aggregation functions that use a new class of parametric aggregator to effectively explore a rich space of possible aggregations. LAF is able to approximate widely used functions as special cases and also has the ability to learn complex functions such as higher-order moments. We evaluated the generalization ability of the framework on synthetic settings as well as real-world datasets, providing comparisons with state-of-the-art sum-decomposition approaches and recently introduced techniques. LAF can be used as off-the-shelf learnable aggregator which uses few parameters and needs little adjustments to be used as aggregation component in any differentiable machine learning model, enhancing the expressivity of architectures and models that deal with unstructured data.

Chapter 5

Applications in recommendation systems

The final goal of a recommendation system is to suggest one or more items that produce positive feedback or response in a user. Recommender systems have been intensively studied in the last decades by the artificial intelligence community, and have become a core component in the industry for their applicability in diverse areas such as e-commerce, search engines, decision aid systems, personalized retail solutions, and advertisement. When a user is faced with a choice, e.g. a movie to watch or a product to buy, the system helps her filter out irrelevant products by discovering and suggesting items that suit her interests. The strategy the recommender system adopts to select the items to propose is indeed the central aspect of such systems. For instance, techniques such as *Collaborative Filtering* [72, 111] recommend items preferred by users similar to the target user, relying on some definition of distance between users. Other techniques based on user information are *Demographic Filtering* techniques [15], in which users are classified according to their demographic information, and the preferences are computed according to the user-user similarity with other users that fell in the same category. In *Content Based Filtering* [87, 22] the proposed objects are extracted by measuring the item-item similarity with other objects the user has liked in the past. Another category of recommendation systems is the *Knowledge Based* [109] systems, which justify their recommendation building an explicit knowledge of the domain, and the *Hybrid Recommender*[20], which combine elements of the different recom-

mendation techniques described above to solve specific recommendation tasks. Moreover, the recommendation can also be performed by gathering aggregate information from sets of entities, as it happens in group recommendation[11, 5] or score prediction[91]. In this section, we explore possible applications of the methods described in Chapter 3 and 4 in recommendation systems.

5.1 Constructive Preference Elicitation Model

In this section we present the core concepts of *Constructive Preference Elicitation* [42], a class of recommendation techniques that combine *Preference Elicitation* [28] procedures with *Constructive Recommendation* [40], and how we can extend these recommenders with the methodologies introduced in this manuscript. Preference Elicitation is the process of modeling and understanding the preferences of a user, or decision-maker, interactively. In this process, the two players, the user and the system, exchange information about the recommended objects until a satisfactory choice is made. The user is responsible for providing feedback on the items recommended by the system. On the other hand, the system's job is to update the user's preference model according to the received feedback and recommend increasing interest items. If the proposed item is not chosen from a set of possible items but instead is configured from scratch by the system, the system solves a Constructive Recommendation problem. Constructive Recommendation techniques aim to synthesize new items or configurations by considering some limitations imposed by the context (modeled as constraints) and the user's preference model. Items are represented as tuples of attributes of different domains, i.e. $x = (x_1, \ldots, x_N)$ where N is the total number of attributes, and each attribute comes from a specific domain, i.e. $x_i \in X_i$ and X is equal to the cartesian product of all the domains. The user's quantitative evaluation of an item can be expressed as a utility function $u(x)$, which assigns a score to the item x. A simple type of utility function is called *Additive Independent* utility [40], which decomposes the overall utility $u(x)$ into the summation of the subutilities $u_i(x_i)$

$$u(x) = \sum_{i=1}^{N} u_i(x_i) \tag{5.1}$$

The recommender's goal is to discover the object to which the user would give the highest possible score among all the available objects. To obtain the optimal recommendation for the user, i.e. x^*, the recommender solves an optimization problem maximizing the utility function's score as follows:

$$x^* = argmax_{x \in X} u(x) \tag{5.2}$$

The preferences of the user can be modeled by using a binary preference relation $\succcurlyeq \in X \times X$, indicating the preference for an item with respect to another item, i.e. $x \succcurlyeq x'$ if item x is "equally preferred or better" than item x'. The utility function expresses the preference of the user in that $x \succcurlyeq x'$ iff $u(x) \geq u(x')$. In the Preference Elicitation process, the preferences' model is iteratively updated until an optimal object is found. During this process, the user is presented with a recommendation, provides feedback on the proposed object, and the system uses this feedback to update its model. A core part of the process is given by the feedback from the user, the fundamental piece of information the system uses to learn the preferences. Several approaches for modeling the user feedback has been proposed. For instance, in the *Pairwise Preference* [48] approach, the user is proposed with two objects, and pick the one that better suits her preferences. Alternatively, in *Set-wise Preference* [113] the user choose the best in a set of objects. Other non-rank based alternatives are *Coactive Learning* [107], in which the user provides as feedback an improved version of the proposed object, or *Coactive Critiquing* [112], where the user specify the reasons why she likes the object or not.

In a Constructive Recommendation System, the object is not chosen but rather synthesized from scratch. In knowledge-based systems terminology, the type of recommendation we exploit lays in the category of Constraint-Based recommendations. The constructive approach we follow is similar to the one presented in *Learning Modulo Theories* [86], a framework for learning and reasoning over complex domains that allows describing the structured object as the combination of several attributes, bounded by some hard constraints that define the space of feasible objects. In a structured prediction task, the goal is to learn a mapping function $f : \mathscr{X} \rightarrow \mathscr{Y}$ between the input space and the structured output space. Examples are given as pairs $(x_i, y_i) \in \mathscr{X} \times \mathscr{Y}$, and the

output object is obtained by solving the following optimization problem

$$f(x) = \arg\max_{y \in \mathscr{Y}} \langle \boldsymbol{w}, \boldsymbol{\phi}(x,y) \rangle \tag{5.3}$$

where $\boldsymbol{w}$ is the parameter vector to be learned from the set of examples, and $\boldsymbol{\phi}(x,y)$ is the feature mapping the pairs into a joint input-output space, $\boldsymbol{\phi} : \mathscr{X} \times \mathscr{Y} \to \mathbb{R}^m$. Therefore, a constructive framework able to create new objects is definitely more flexible than a recommendation system that simply selects an object from a given set of possible solutions. A notable implementation of a Constructive Preference Elicitation model in a real world scenario is called *Smart Plan* [41]. Smart Plan constructs bundles of telecommunication products (called "plans") tailored to the user interest, which provides her feedback via Coactive Learning. This work represents the first application of this type, showing how these type of interactive approaches to recommendation can significantly improve user satisfaction while maintaining the cognitive cost of discovering the optimal solution low.

5.1.1 Extension with Relational Metrics

The feature definition and the relational metric can be employed to help a constructive recommender to discover more easily the preferences of a user. Constructive recommendation is a content based approach that during the initial phase it could suffer of a cold start, i.e., recommendations at the beginning of the process might be not so relevant, due to the little information of the user's preferences. In the following part we give the description of an Hybrid recommender system, that combine the content based approach of constructive recommendation with a collaborative filtering approach we exploit using the TET relational metric introduced in Chapter 3. We add to the utility function a component which weights the score of the utility function of other users u' to an object x. Consider a user u as an entity of a relational domain. Therefore, we can construct a TET that models the relational feature of u and, by applying the logistic evaluation function, extract the logistic evaluation tree L_u. We also denote a generic TET-based distance function as $d(L,L')$. We define the utility score for an object x as a linear combination of the user's utility function and the utility scores of other users on x. The contribution of each user u' in the

relational domain to the score of $f_u(x)$ is weighted by the similarity of user u with user u', defined as follow:

$$f_u(x) = \boldsymbol{w}_u^T \boldsymbol{\phi}(x) + \gamma \sum_{i=1}^{j} \frac{\boldsymbol{w}_i^T \boldsymbol{\phi}(x)}{d(L_u, L_i)} \tag{5.4}$$

If we rewrite the above equation by grouping the feature representation of the object x, the equation becomes

$$f_u(x) = \left(\gamma \sum_{i=1}^{j} \frac{\boldsymbol{w}_i}{d(L_u, L_i)} + \boldsymbol{w}_u \right)^T \boldsymbol{\phi}(x) \tag{5.5}$$

where γ is an additional parameter we introduce to tune the overall contribution of other users w.r.t. the base utility of u, and $d(L, L')$ is a distance value between two users. The collaborative filtering component in the formulation is pretty clear, as we integrate the distance between users to determine the preference model. The role of the weight γ is crucial during the preference elicitation process. If the user u is a new user in the system, the value of γ should be high, in fact the system doesn't have much information about the preferences of u, so recommendations are mostly based on the preferences of similar users. However, the contributions of other users decay as the system learns the preferences of u, thus also γ value decrease. Finally, to recommend a new object to the user based on her preferences, we maximize the score of the utility function.

5.2 Group Recommendation

The challenge faced by a recommendation system is usually to suggest the best option to a single user. However, on many occasions, users participate collectively to decide, for instance, which movie to watch, list of songs to listen to, in which restaurant to eat or to plan the next travel, arising a new set of recommendation problems in which the recommendation should consider all the group members' preferences, not only one user. On a different scale, also in social media users naturally form groups of people who share common interests, beliefs, political views or simply discuss about some arguments, or just for having fun with some memes [71]. Examples of these groups are, for instance, users that follow the same page on Facebook [3], or the same subreddit in Reddit, or users that listen to the same playlist on Spotify. Identifying and understanding this group's behavior has been a relevant topic in the recommendation systems community, referring to the task of modeling the preferences of multiple users all together as *Group Recommendation*. The main challenge of a group recommendation system is to model the users' preferences in an effective way such that the newly obtained preferences reflect the characteristics of the entire group. Groups are usually defined as homogeneous, i.e. group members share common interests, or heterogeneous. There are two main approaches to model group recommendations. In the first strategy the group profile is created by aggregating the individual preferences of the users into a single representation, and then the items are recommended based on this aggregated profile [66]. The second strategy is to generate the recommendations for the members and then aggregating them into a single group recommendation [65]. In the first strategy the aggregation step happens before the recommendation, while in the second the recommendation precedes the aggregation. Some of the most used strategies to aggregate come from the *group decision making theory* [29, 79]. Some functions aggregate information from only a subset of group members, e.g. in Least Misery the recommended items is the one that obtains the highest score among all lowest evaluations of the users, or in the case of the Most Respected Person, in which the evaluation is selected based on the most "important" user in the group. Functions that consider all the members equally are called *democratic functions*. These can also be quite sofisticated functions, but they rely on some aggregations functions such as the average or the sum of

the users evaluations to select an item. Methods like Average Without Misery, Multiplicative, Additive Utilitarian belong to this class. The choice of this aggregation strategies influence heavily the performance of the recommendation system. Therefore, a strategy to directly learn the aggregation while optimizing the recommendations could be beneficial for this type of systems. In this context, LAF (Chapter 4) could be applied as substitute for the strategies presented above. Consider an evaluation function that determines the preference score a user u would give to an item t in the form $e_u = e(u,t)$. If s has an upper bound on the produced score, e.g. the star scoring system for products in e-commerce or for movies, then we can normalize this value to lay in the range $[0,1]$. Therefore, we can collect this evaluations of a group G for an item t as the set $\boldsymbol{e} = \{e_{u_1}, \ldots, e_{u_n}\}$ composed of all the single evaluations made by the users:

$$s_{G,t} = LAF(\boldsymbol{e}) \tag{5.6}$$

By performing this operation for each item, the system can list the items in decreasing order based on the calculated s score, obtaining a ranking of the items for the group. In order for LAF to be applied in this context, some sort of supervision is needed, therefore the use of a loss function to measure how much the proposed rank of items differ from the perfect one. There are several differentiable ranking loss that can be employed in this case, such as Bayesian Personalized Ranking [24] or other pairwise loss functions [23].

Chapter 6

Conclusions

This manuscript introduced several new methodologies to learn on relational structures and provided some possible applications of the proposed methods in the recommendation systems domain. Specifically, we introduce a new class of metrics that exploit counts-of-counts statistics to determine the similarity between two entities in a relational graph. This framework allows to combine attributes of the entities with the attributes and the structural properties of their relational neighborhood. We showed how this method outperforms other solutions based on graphs kernel in classification and regression tasks, and its capability in the task of information retrieval. Additionally, we showed how performing metric learning on this metric can improve the performances of a KNN classifier to perform competitively with neural architectures for relational data. An interesting research direction would be to convert the distance function into a similarity function by transforming it into a kernel, and applying it in other models such as SVM. In the second research work, we defined a new learnable aggregator, which includes as special cases some widely used aggregation functions, while learning more complex functions as higher moments. We provided an extensive experimental evaluation on some graph tasks and set learning tasks, compared with recent solutions and state-of-the-art models, outperforming them in most of the tasks. The flexibility of the model combined with its expressiveness makes it an exceptional candidate that can be employed in any end-to-end differentiable model due to its flexibility and easy applicability. In this sense, the most promising research direction would be to employ LAF into different models, to assess its expressiveness and learning ability. Finally, we proposed an extension to a recommendation system based

on Constructive Preference Elicitation that uses the relational metrics to define the prior preferences of a user in a relational domain. We also proposed LAF as possible alterative to aggregate information in Group Recommendation Systems.

References

[1] Alshehhi, R. and Marpu, P. R. (2017). Hierarchical graph-based segmentation for extracting road networks from high-resolution satellite images. *ISPRS journal of photogrammetry and remote sensing*, 126:245–260.

[2] Amatriain, X. and Basilico, J. (2015). Recommender systems in industry: A netflix case study. In *Recommender systems handbook*, pages 385–419. Springer.

[3] Baatarjav, E.-A., Phithakkitnukoon, S., and Dantu, R. (2008). Group recommendation system for facebook. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 211–219. Springer.

[4] Babai, L. and Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE.

[5] Baltrunas, L., Makcinskas, T., and Ricci, F. (2010). Group recommendations with rank aggregation and collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 119–126.

[6] Barla, A., Odone, F., and Verri, A. (2003). Histogram intersection kernel for image classification. In *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, volume 3, pages III–513–16 vol.2.

[7] Bellet, A., Habrard, A., and Sebban, M. (2012). Similarity learning for provably accurate sparse linear classification. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12.

[8] Bellet, A., Habrard, A., and Sebban, M. (2013a). A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709.

[9] Bellet, A., Habrard, A., and Sebban, M. (2013b). A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*.

[10] Ben-Hur, A. and Noble, W. S. (2005). Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl_1):i38–i46.

[11] Berkovsky, S. and Freyne, J. (2010). Group-based recipe recommendations: analysis of data aggregation strategies. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 111–118.

[12] Bianchini, M. and Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565.

[13] Bilenko, M., Basu, S., and Mooney, R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 11.

[14] Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1):285 – 297.

[15] Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, 46:109 – 132.

[16] Borgwardt, K. M. and Kriegel, H.-P. (2005). Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE.

[17] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

[18] Boyer, L., Habrard, A., and Sebban, M. (2007). Learning metrics between tree structured data: Application to image recognition. In *Machine Learning: ECML 2007, 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007, Proceedings*, pages 54–66.

[19] Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*.

[20] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370.

[21] Campbell, C. (2002). Kernel methods: a survey of current techniques. *Neurocomputing*, 48(1):63 – 84.

[22] Cantador, I., Bellogín, A., and Vallet, D. (2010). Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 237–240. ACM.

[23] Cao, D., He, X., Miao, L., An, Y., Yang, C., and Hong, R. (2018). Attentive group recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 645–654.

[24] Cao, D., Nie, L., He, X., Wei, X., Zhu, S., and Chua, T.-S. (2017). Embedding factorization models for jointly recommending items and user generated lists. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 585–594.

[25] Chan, T., Esedoglu, S., and Ni, K. (2007). Histogram based segmentation using Wasserstein distances. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 697–708. Springer.

[26] Chatpatanasiri, R., Korsrilabutr, T., Tangchanachaianan, P., and Kijsirikul, B. (2010). A new kernelization framework for mahalanobis distance learning algorithms. *Neurocomputing*, 73(10-12):1570–1579.

[27] Chechik, G., Shalit, U., Sharma, V., and Bengio, S. (2009). An online algorithm for large scale image similarity learning. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 306–314.

[28] Chen, L. and Pu, P. (2004). Survey of preference elicitation methods. Technical report.

[29] Chevaleyre, Y., Endriss, U., Lang, J., and Maudet, N. (2007). A short introduction to computational social choice. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 51–69. Springer.

[30] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, CVPR '05, pages 539–546.

[31] Clarkson, K. L. (2006). Nearest-neighbor searching and metric space dimensions. In *In Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*. MIT Press.

[32] Codd, E. F. (2002). A relational model of data for large shared data banks. In *Software pioneers*, pages 263–294. Springer.

[33] Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*.

[34] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[35] Cui, Y., Zhou, F., Lin, Y., and Belongie, S. (2016). Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1153–1162.

[36] Datta, R., Joshi, D., Li, J., and Wang, J. Z. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):5:1–5:60.

[37] Davis, J. V., Kulis, B., Jain, P., Sra, S., and Dhillon, I. S. (2007). Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07.

[38] De Raedt, L. and Kersting, K. (2008). Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, pages 1–27. Springer.

[39] Dhillon, P. S., Talukdar, P. P., and Crammer, K. (2010). Inference-driven metric learning for graph construction. In *4th North East Student Colloquium on Artificial Intelligence*.

[40] Dragone, P. (2017). Constructive recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 441–445, New York, NY, USA. Association for Computing Machinery.

[41] Dragone, P., Pellegrini, G., Vescovi, M., Tentori, K., and Passerini, A. (2018a). No more ready-made deals: constructive recommendation for telco service bundling. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 163–171. ACM.

[42] Dragone, P., Teso, S., and Passerini, A. (2018b). Constructive preference elicitation. *Frontiers in Robotics and AI*, 4:71.

[43] Egghe, L. (2006). Theory and practise of the g-index. *Scientometrics*, 69(1):131–152.

[44] Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. (2019). Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426.

[45] Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., and Borgwardt, K. (2013). Scalable kernels for graphs with continuous attributes. *Advances in neural information processing systems*, 26:216–224.

[46] Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. (2018). Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877.

[47] Filippone, M., Camastra, F., Masulli, F., and Rovetta, S. (2008). A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176 – 190.

[48] Fürnkranz, J. and Hüllermeier, E. (2003). Pairwise preference learning and ranking. In *European conference on machine learning*, pages 145–156. Springer.

[49] Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, volume 174. freeman San Francisco.

[50] Ghahramani, Z. and Heller, K. A. (2006). Bayesian sets. In *Advances in neural information processing systems*, pages 435–442.

[51] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

[52] Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

[53] Hamilton, W. L., Ying, Z., and Leskovec, J. (2017b). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1024–1034.

[54] Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz.

[55] Hirsch, J. E. (2005). An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences of the United States of America*, 102(46):16569.

[56] Hoi, S. C., Liu, W., Lyu, M. R., and Ma, W.-Y. (2006). Learning distance metrics with contextual constraints for image retrieval. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2072–2078. IEEE.

[57] Ilse, M., Tomczak, J. M., and Welling, M. (2018). Attention-based deep multiple instance learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2132–2141.

[58] Jaeger, M. (2006). Type extension trees: a unified framework for relational feature construction. In *Proceedings of Mining and Learning with Graphs (MLG-06)*.

[59] Jaeger, M. (2007). Parameter learning for relational bayesian networks. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 369–376, New York, NY, USA. Association for Computing Machinery.

[60] Jaeger, M., Lippi, M., Passerini, A., and Frasconi, P. (2013). Type extension trees for feature construction and learning in relational domains. *Artificial Intelligence*, 204:30–55.

[61] Jaeger, M., Lippi, M., Pellegrini, G., and Passerini, A. (2019). Counts-of-counts similarity for prediction and search in relational data. *Data Mining and Knowledge Discovery*, pages 1–44.

[62] Kang, U., Tong, H., and Sun, J. (2012). Fast random walk graph kernel. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 828–838. SIAM.

[63] Kazemi, S. M. and Poole, D. (2018). RelNN: A deep neural model for relational learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[64] Kedem, D., Tyree, S., Sha, F., Lanckriet, G. R., and Weinberger, K. Q. (2012). Non-linear metric learning. In *Advances in Neural Information Processing Systems 25*, pages 2573–2581.

[65] Kim, H.-N. and El Saddik, A. (2015). A stochastic approach to group recommendations in social media systems. *Information Systems*, 50:76–93.

[66] Kim, J. K., Kim, H. K., Oh, H. Y., and Ryu, Y. U. (2010). A group recommendation system for online communities. *International journal of information management*, 30(3):212–219.

[67] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

[68] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

[69] Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

[70] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

[71] Kompan, M. and Bielikova, M. (2014). Group recommendations: Survey and perspectives. *Computing and Informatics*, 33(2):446–476.

[72] Koren, Y. and Bell, R. (2015). *Advances in Collaborative Filtering*, pages 77–118. Springer US, Boston, MA.

[73] Krishna, S. (1992). *Introduction to database and knowledge-base systems*, volume 28. World Scientific.

[74] Lee, J., Lee, Y., Kim, J., Kosiorek, A. R., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3744–3753.

[75] Lippi, M., Jaeger, M., Frasconi, P., and Passerini, A. (2011). Relational information gain. *Machine Learning*, 83:219–239.

[76] Liu, T., Moore, A. W., Yang, K., and Gray, A. G. (2005). An investigation of practical approximate nearest neighbor algorithms. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 825–832. MIT Press.

[77] Ljosa, V., Bhattacharya, A., and Singh, A. K. (2006). Indexing spatially sensitive distance measures using multi-resolution lower bounds. In *International Conference on Extending Database Technology*, pages 865–883. Springer.

[78] Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011). Recommender systems with social regularization. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 287–296, New York, NY, USA. ACM.

[79] Masthoff, J. (2011). Group recommender systems: Combining individual models. In *Recommender systems handbook*, pages 677–702. Springer.

[80] Melnikov, V. and Hüllermeier, E. (2016). Learning to aggregate using uninorms. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II*, pages 756–771.

[81] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[82] Moschitti, A. (2006). Making tree kernels practical for natural language learning. In *11th conference of the European Chapter of the Association for Computational Linguistics*.

[83] Muja, M. and Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240.

[84] Neuhaus, M. and Bunke, H. (2007). Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239–247.

[85] Orsini, F., Frasconi, P., and De Raedt, L. (2015). Graph invariant kernels. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence*, volume 2015, pages 3756–3762. IJCAI-INT JOINT CONF ARTIF INTELL.

[86] Passerini, A. (2016). Learning modulo theories. In *Data Mining and Constraint Programming*, pages 113–146. Springer.

[87] Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer.

[88] Pele, O. and Werman, M. (2008). A linear time histogram metric for improved SIFT matching. In Forsyth, D. A., Torr, P. H. S., and Zisserman, A., editors, *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III*, volume 5304 of *Lecture Notes in Computer Science*, pages 495–508. Springer.

[89] Pellegrini, G., Tibo, A., Frasconi, P., Passerini, A., and Jaeger, M. (2020). Learning aggregation functions. *arXiv preprint arXiv:2012.08482*.

[90] Qamar, A. M., Gaussier, E., Chevallet, J., and Lim, J. H. (2008). Similarity learning for nearest neighbor classification. In *2008 Eighth IEEE International Conference on Data Mining*.

[91] Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer.

[92] Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62(1):107–136.

[93] Ristad, E. S. and Yianilos, P. N. (1998). Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532.

[94] Ristoski, P. and Paulheim, H. (2016). Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer.

[95] Rubner, Y., Tomasi, C., and Guibas, L. J. (1998). A metric for distributions with applications to image databases. In *Computer Vision, 1998. Sixth International Conference on*, pages 59–66. IEEE.

[96] Rusu, R. B. and Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE.

[97] Salakhutdinov, R. and Hinton, G. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. In Meila, M. and Shen, X., editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2, pages 412–419.

[98] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

[99] Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1):115–153.

[100] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer.

[101] Schölkopf, B. (2001). The kernel trick for distances. In *Advances in neural information processing systems*, pages 301–307.

[102] Schölkopf, B. and Smola, A. (2002). *Learning with Kernels*. The MIT Press, Cambridge, MA.

[103] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[104] Schultz, M. and Joachims, T. (2004a). Learning a distance metric from relative comparisons. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, pages 41–48.

[105] Schultz, M. and Joachims, T. (2004b). Learning a distance metric from relative comparisons. *Advances in neural information processing systems*, 16:41–48.

[106] Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

[107] Shivaswamy, P. and Joachims, T. (2015). Coactive learning. *Journal of Artificial Intelligence Research*, 53:1–40.

[108] Silva, N. B., Tsang, R., Cavalcanti, G. D., and Tsang, J. (2010). A graph-based friend recommendation system using genetic algorithm. In *IEEE congress on evolutionary computation*, pages 1–7. IEEE.

[109] Smyth, B. (2007). *Case-Based Recommendation*, pages 342–376. Springer Berlin Heidelberg, Berlin, Heidelberg.

[110] Stark, C., Breitkreutz, B.-J., Reguly, T., Boucher, L., Breitkreutz, A., and Tyers, M. (2006). Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl_1):D535–D539.

[111] Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

[112] Teso, S., Dragone, P., and Passerini, A. (2017). Coactive critiquing: Elicitation of preferences and features. In *AAAI*, pages 2639–2645.

[113] Teso, S., Passerini, A., and Viappiani, P. (2016). Constructive preference elicitation by setwise max-margin learning. *arXiv preprint arXiv:1604.06020*.

[114] Tibo, A., Frasconi, P., and Jaeger, M. (2017). A network architecture for multi-multi-instance learning. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part I*, pages 737–752.

[115] Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., and Borgwardt, K. (2019). Wasserstein weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems*, pages 6439–6449.

[116] Uhlmann, J. K. (1991). Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179.

[117] Van Assche, A., Vens, C., Blockeel, H., and Džeroski, S. (2006). First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning*, 64(1-3):149–182.

[118] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[119] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *ICLR'18*.

[120] Wagstaff, E., Fuchs, F. B., Engelcke, M., Posner, I., and Osborne, M. (2019). On the limitations of representing functions on sets. *arXiv preprint arXiv:1901.09006*.

[121] Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for similarity search: A survey. *CoRR*, abs/1408.2927.

[122] Wang, J., Zhang, T., Song, J., Sebe, N., and Shen, H. T. (2017). A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1.

[123] Weisfeiler, B. and Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16.

[124] Wu, L., Jin, R., Hoi, S. C., Zhu, J., and Yu, N. (2009). Learning bregman distance functions and its application for semi-supervised clustering. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*.

[125] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920.

[126] Xing, E. P., Ng, A. Y., Jordan, M. I., and Russell, S. (2002). Distance metric learning, with application to clustering with side-information. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*.

[127] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

[128] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3391–3401. Curran Associates, Inc.

[129] Zesch, T., Müller, C., and Gurevych, I. (2008). Extracting lexical semantic knowledge from wikipedia and wiktionary. In *LREC*, volume 8, pages 1646–1652.

[130] Zhang, C.-T. (2009). The e-index, complementing the h-index for excess citations. *PLoS One*, 4(5):e5429.

[131] Zhou, G., Zhang, M., Ji, D., and Zhu, Q. (2007). Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 728–736.