Graph-based semi-supervised learning in missing and noisy graph settings

Mariana Vargas Vieyra

Apprentissage Semi-supervisé Basé sur les Graphes Avec des Graphes Manquants et Bruités

Graph-based Semi-supervised Learning in Missing and Noisy Graph Settings

Mariana Vargas Vieyra

Thèse preparée au sein de l'équipe Magnet, Inria, et CRIStAL, Université de Lille, sous la direction de Dr. Marc Tommasi et l'encadrement de Dr. Pascal Denis et Dr. Aurélien Bellet.

Soutenue publiquement le 27 octobre 2021 devant le jury:

M. Eric Gaussier	Professeur, Université Grenoble Alpes	Rapporteur
Mme. Céline Hudelot	Professeure, Ecole Centrale Paris	Rapportrice
M. Michalis Vazirgiannis	Professeur, Ecole Polytechnique	Examinateur
M. Pascal Denis	Chargé de Recherche, Inria	Encadrant
M. Aurélien Bellet	Chargé de Recherche, Inria	Co-encadrant
M. Marc Tommasi	Professeur, Université de Lille	Directeur

En vue de l'obtention du titre de
Docteur en Informatique

To my parents Piqui and José
To my grandmother Dorita
To my grandmother "La Gringa", in loving memory

Acknowledgements

First and foremost, I would like to thank my advisors Pascal Denis and Aurélien Bellet. Since the first day you provided me with your constant support and guidance, steering my research in the right direction while allowing me to propose and develop my own ideas. I remain indebted to you for your dedication, encouragement and patience.

I would also like to express my deepest appreciation to my thesis committee members for reviewing my work, and for the challenging questions and interesting discussion that took place during my defense.

During my PhD I had the pleasure and privilege to interact with a group of wonderful people: researchers, professors, engineers and other fellow students. I thank you all for your friendship and comradeship, and for all the good moments spent together. I'd also like to extend my gratitude to all my "lillois" friends, whom I was very lucky to meet in the past four years, and those friends who always remained close to me despite being on the other side of the world.

My PhD has been one of the most exciting, interesting, and yet challenging periods of my life, and I wouldn't have made it through without the unconditional love and care of my family. I especially thank my brothers and my sister, my grandmother, and my parents, to whom I owe more than words could ever express.

Abstract

In the last few years Machine Learning methods have been incorporated in various Natural Language Processing systems. As a result, these methods have shown impressive results in a variety of tasks across multiple domains, in particular, through supervised learning. However, these methods usually rely on large amounts of labeled data, implying a strong presence of human intervention in the modeling pipeline and a potential high cost for data annotation. Graph-based Semi-supervised Learning (GSSL) is a framework that alleviates these issues by exploiting the information provided by the unlabeled data. It takes as input a dataset and a graph that represents pairwise connections between elements, both labeled and unlabeled. A bottleneck in the use of GSSL in arbitrary datasets is that a graph is not always readily available, and although there are heuristic techniques to build them, they usually fall short of capturing the true topology of the data.

In this thesis we propose two original methods to deal with scenarios where labeled data is scarce and where either no graph is available, or where the *a-priori* graph is considered a noisy observation of an unknown true graph. Our first method combines Graph Learning and Metric Learning to jointly learn a graph and a data transformation that we can subsequently plug into a standard GSSL algorithm such as Label Spreading of Graph Convolutional Networks. For our second method we adopt a probabilistic approach and use the tools from deep generative models to build a framework where we jointly infer a graph and the parameters of a semi-supervised classification model in an end-to-end fashion. We empirically show that our methods yield competitive results in text classification. Furthermore, we are able to learn task-specific graphs that capture interesting properties about the data. Finally, we identify challenges and discuss potential directions to address them.

Resumé

Au cours des dernières années, les méthodes d'apprentissage automatique ont été intégrées dans divers systèmes de traitement du langage naturel. Ces méthodes ont montré des résultats impressionnants dans une variété de tâches dans de multiples domaines, en particulier par l'apprentissage supervisé. Cependant, ces méthodes reposent généralement sur de grandes quantités de données étiquetées, ce qui implique une forte intervention humaine dans le pipeline de modélisation et un coût potentiel élevé pour l'annotation des données. L'apprentissage semi-supervisé basé sur les graphes (GSSL) est un cadre théorique qui atténue ces problèmes en exploitant les informations fournies par les données non étiquetées. Il prend en entrée un ensemble de données et un graphe qui représente les connexions entre les éléments, étiquetés et non étiquetés. Un obstacle dans l'utilisation de GSSL est qu'un graphe n'est pas toujours disponible, et bien qu'il existe des techniques heuristiques pour les construire, elles ne parviennent généralement pas à capturer la véritable topologie des données.

Dans cette thèse, nous proposons deux méthodes originales pour traiter les scénarios où les données étiquetées sont rares et où le graphe n'est disponible ou est seulement une observation bruitée d'un vrai graphe inconnu. Notre première méthode combine l'apprentissage des graphes et l'apprentissage des métriques pour apprendre conjointement un graphe et une transformation de données que nous pouvons ensuite insérer dans un algorithme GSSL standard, comme par exemple Label Spreading ou Graph Convolutional Networks. Pour notre deuxième méthode, nous adoptons une approche probabiliste et utilisons les outils des modèles génératifs pour construire un cadre dans lequel nous inférons conjointement un graphe et les paramètres d'un modèle de classification semi-supervisée "end-to-end". Nous montrons empiriquement que nos méthodes donnent des résultats compétitifs dans la classification de textes. De plus, nous obtenons des graphes spécifiques aux tâches qui capturent des propriétés intéressantes sur les données. Finalement, nous identifions les défis et discutons des directions potentielles pour les relever.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In past few decades we have witnessed a dramatic improvement of Natural Language Processing (NLP) systems thanks to the incorporation of Machine Learning (ML) tools in problems so far addressed with symbolic methods. In particular, with the advent of automatic differentiation, it became possible to design and deploy complex deep learning architectures, and incorporate them in the NLP pipeline. For example, neural models for learning word representations or "embeddings" [48, 53, 17] became a building block in many NLP systems. As a result, many tasks such as part of speech (POS) tagging [2], document classification [1], and machine translation [10] saw unprecedented performance.

However, this new NLP paradigm brought along new challenges. Modern NLP systems require large amounts of annotated data in order to train models, which represents a bottleneck in practice. Moreover, one usually needs access to annotated data belonging to the domain of interest for NLP systems to have a good performance. A paradigmatic example is the performance gap between NLP models for resourceful languages like English, and low resources languages where models are trained on relatively small corpora. Even in the case where we have access to a large corpus of text, it is a known fact that the annotation process can be very costly and time consuming [20]. On the other hand it is the case that unannotated data is plentiful for many applications. A relevant question in this context is, *can we build systems that require less human intervention, and that exploit information from unannotated data?* In this thesis we argue that Graph-based Machine Learning is an adequate framework to addressing both problems.

Graph-based ML [61] exploits prior knowledge that comes in the form of a graph that encodes information about how the elements of interest interact. Nodes and edges in a graph can represent many different things. For instance, citation networks can be deemed a graph where nodes are scientific articles and edges connect two nodes when one article cites the other. Other examples are social networks where nodes are people and edges represent

whether two people are friends, or parsing trees where nodes are words in a sentence and edges represent a syntactic relationship between them. In all these examples the graph is naturally provided by the data, and informs about the data distribution. Exploiting this information is known to improve the classification performance of some tasks [77, 74, 39]. To illustrate this, let us take the citation network example we mentioned above and let us imagine the following scenario. We have a set of scientific papers classified into topics, and a citation graph that connects two papers if one cites the other. We have a small training set of labeled articles and the task is to find the labels of the unlabeled set. Since labeled data is scarce, supervised methods will tend to quickly overfit. In contrast, citations between articles convey relevant information that complement the textual content, and exploiting this can improve over supervised models.

Graph-based Semi-supervised Learning (GSSL) is an area of Machine Learning that studies semi-supervised learning algorithms where the structure of the data is represented by a graph. This area has seen some recent successes. Notably, the method introduced by Kipf and Welling [39] based on Graph Convolutional Networks achieved significant performance gains over its predecessors.

A problem arises when a graph is not readily available, or when it represents a noisy observation of the true graph. Classical GSSL methods do not account for graph uncertainty and therefore, it is not straightforward to use this framework in scenarios where we do not have a graph or where we can not fully rely on the graph. When unavailable, we can use heuristics to construct a graph using the data. For example, one can connect each element in the training set with its k nearest neighbors (kNN graph). Alternatively, one can create a graph where edges are assigned a weight that is inversely proportional to the Euclidean distance between the points. Although using GSSL with these heuristic structures can improve over supervised methods, these graphs are usually not the optimal options.

There is a vast variety of approaches concerned with constructing a graph in a data-driven manner. Some of these methods are unsupervised and thus disregard the label information, and others take into account the supervised information but fail to fully capture the geometry of the data. We will discuss many of these methods in the next chapter. In general, *the problem of performing GSSL in unavailable/unreliable graph scenarios is arguably under-explored, and it is the purpose of this thesis to contribute to filling that gap*. We then propose algorithms where the graph is built in a task-driven manner. We also explore the possibility of tackling two problems simultaneously, that is, performing graph inference and GSSL at the same time. In this work we present the following contributions.

(i) We introduce a method where we learn a good representation of the data and a graph simultaneously. The data representation aims at pulling together data points that are

likely to belong to the same class, while pushing apart those that are likely to have different class memberships. Also, it guides the graph learning process towards graphs that encode the geometry of the transformed data. This way we obtain a task-specific graph and a data representation that we can plug into a GSSL algorithm. Through the hyperparameters of the objective function we can control the sparsity of the resulting graph.

(ii) Taking a step further, we propose an end-to-end pipeline where the graph is considered to be a set of latent variables, and where we perform inference in the graph and the parameters of a classification model simultaneously. This model is more flexible than the previous one in the sense that, by using a Bayesian framework, we have control over the prior of the graph. In particular, we can assign a higher prior probability to some edges, thus encoding a prior graph preference. We demonstrate that our method outperforms supervised and semi-supervised baselines where the graph is heuristically computed, and that reaches state-of-the-art results.

1.1 Published Work

The contributions of this thesis have resulted in two publications:

(i) Mariana Vargas Vieyra, Aurélien Bellet, Pascal Denis, 2020, *Joint Learning of the Graph and the Data Representation for Graph-Based Semi-Supervised Learning*, Proceedings of the Graph-based Methods for Natural Language Processing (TextGraphs).

(ii) Mariana Vargas Vieyra, Aurélien Bellet, Pascal Denis, 2019, *Probabilistic End-to-End Graph-based Semi-Supervised Learning*, contributed talk at NeurIPS Graph Representation Learning workshop, poster at NeurIPS Bayesian Deep Learning workshop.

1.2 Outline

We structure this thesis in five chapters.

Chapter 2: Background We provide details about the frameworks that are relevant to this thesis. We discuss graphs and their importance in Machine Learning, some methods to construct them, and a framework for dealing with unobserved random variables that is useful for the unavailable/unreliable graph scenario we are interested in.

Chapter 3: Joint Learning of the Graph and the Data Representation We present our first contribution. We describe an algorithm that learns a graph and a data representation jointly. We present an optimization scheme where we alternate between two subproblems: fixing the graph and optimizing the data representation, and keeping the data representation fixed to optimize the graph. We provide empirical evidence that shows that our graph and data representation outperform heuristic graphs.

Chapter 4: Graph Inference and SSL with Auto-Encoding Variational Bayes We take a step further and describe an end-to-end system that performs inference in the graph and GSSL simultaneously. We use a Bayesian approach to account for graph uncertainty, and present a model where the graph is considered a set of independent unobserved random variables. We describe an optimization procedure based on recent advances in Variational Inference.

Chapter 5: Conclusion In this chapter we summarize our contributions and provide an outlook for future directions.

Chapter 2

Background

In this chapter we describe the grounds upon which this work is built. Additionally, we characterize the task of Graph-based Semi-supervised Learning in scenarios where the graph is unavailable and motivate the use of specific frameworks and tools to address it.

We will start by providing an overview of graphs and Graph Signal Processing. We are interested in how graphs can represent a domain of data that is non-Euclidean, allowing to generalize notions from classic calculus to more complex and realistic domains. In particular, graphs carry geometric information about the data they are associated with, which is very helpful in applications where we have limited access to supervised information. In this context, where regular supervised learning methods would generalize poorly to unseen data, the graph can guide the learning process to solutions that do not overfit. Algorithms that follow this principle are considered in the area of Graph-based Semi-Supervised Learning (GSSL). The main question GSSL addresses is, given a graph where some of the nodes are associated with observations from some unknown labeling function, how to find the missing labels for the rest of the nodes?

Despite the success of GSSL, graphs are not always readily available. In fact, very often one has to construct it following some heuristic that indicates "similarity" or "dissimilarity" between points. These heuristics for graph construction can seem rather arbitrary, and in fact may lead to suboptimal results. The reason for this is that it is no easy task to find the graph that best fits a dataset. For example, the Euclidean distance may not capture important relationships between features. As a matter of fact, finding a graph that captures the geometry of the data is a research question on its own, studied in the field of Graph Learning.

Better graphs can be constructed if we have a more appropriate notion of similarity. More specifically, one could learn a metric function, other than the Euclidean distance, that better explains the underlying notion of similarity in the task of interest. The area of Machine Learning that deals with finding such metric functions in a data-driven fashion is Metric

Learning. Roughly speaking, these methods will incorporate prior knowledge about the data in the form of similar/dissimilar restrictions, and will exploit that information in order to generalize to a metric function.

Another possibility when there is uncertainty about the graph is to adopt a Bayesian approach. In this framework we can consider the graph to be an unobserved random variable, or *latent variable*, with some distribution we choose. On the one hand, to pick a family of distributions for the graph is a way of incorporating our prior knowledge about the structure of the data, on the other hand, the uncertainty about the graph can be accounted for in Bayesian inference.

We will review all the above concepts in the rest of this Chapter. In Section 2.1 we will provide an overview of graphs and GSP, in Section 2.2 we present Graph-based Semi-supervised Learning and some popular algorithms. Metric Learning will be discussed in Section 2.3, and Latent Variable Models in Section 2.4. Finally, in Section 2.5 we introduce Graph Learning and its role in GSSL.

Contents

2.1 Graphs

In this section we are going to introduce one of the building blocks of this thesis: graphs. Roughly speaking, graphs are a very expressive data structure that capture relationships between arbitrary objects. They have been widely used in a vast variety of contexts and applications, in particular, in Machine Learning and Statistics, where they can have an interpretation in terms of the independence structure of a set of random variables, interaction of dynamic systems, and knowledge graphs, just to name a few examples.

In what follows we are going to narrow down the topic of graphs to what is relevant for this work. That is, first, we will formally define what graphs are and describe some of their main properties, then, we are going to discuss the notion of *graph spectrum*, and finally, their role in Signal Processing and how graphs can be used to reason about data generated from domains that are non-Euclidean.

2.1.1 Spectral Graph Theory

Graphs, in a broad sense, are mathematical objects that represent pairwise relationships between elements in a set. They are usually associated with matrices that carry important information about the task of interest. The field of *Spectral Graph Theory* is concerned with the study of certain properties of the graph and its associated matrices. But before delving into this concepts let us introduce some basic definitions.

Definition 1. *A graph is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V}$ is a set of elements we call vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set that encodes pairwise relationships between elements in $\mathcal{V}$. A pair $(v_i, v_j) \in \mathcal{E}$ is called an edge. A graph $\mathcal{G}$ is said to be a valued graph or weighted graph when it is endowed with a weight function $\omega : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$ such that $\omega(v_i, v_j) > 0$ if $(v_i, v_j) \in \mathcal{E}$, $\omega(v_i, v_j) = 0$ otherwise. We denote such a graph with the triplet $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$.*

Observe that a non-weighted graph is a special case of a weighted graph with binary values. In the following definition we introduce a compact notation for the weight function.

Definition 2. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$ be a graph. Let $n = |\mathcal{V}|$ be the size of $\mathcal{V}$. The function ω can be represented by a weight matrix $W \in \mathbb{R}^{n \times n}$ such that*

$$W_{ij} = \begin{cases} \omega(v_i, v_j) & (v_i, v_j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases}$$

We call such matrix the adjacency matrix of $\mathcal{G}$.

As a data structure graphs are very expressive and can naturally arise in many real-world contexts. For example, in social networks users are nodes in a graph and an edge connects two users when one befriends another. In circuit networks edges represent wires between electronic components, which are nodes, or in biology, where graphs are used to encode interactions between proteins [69]. Figure 2.1 shows an example of a graph that arises from a random sensor network with 100 nodes and 469 edges generated with the PyGSP Python package . A classical example in NLP is that of citation networks: we have a set of scientific articles, each of which represents a node in a graph, and an edge will connect two articles if one cites the other. Cora [58] and Citeseer [42] are two popular citation network datasets. Figure 2.2 shows a visualization of these graphs.

We say the graph is *undirected* if whenever $(v_i, v_j) \in \mathcal{E}$ then $(v_j, v_i) \in \mathcal{E}$, and if $\omega(v_i, v_j) = \omega(v_j, v_i)$. Otherwise, we say the graph is *directed*. In the following, unless otherwise stated, we will consider undirected graphs.

We can now introduce the notion of *graph Laplacian*, a concept that will be of crucial importance throughout this work.

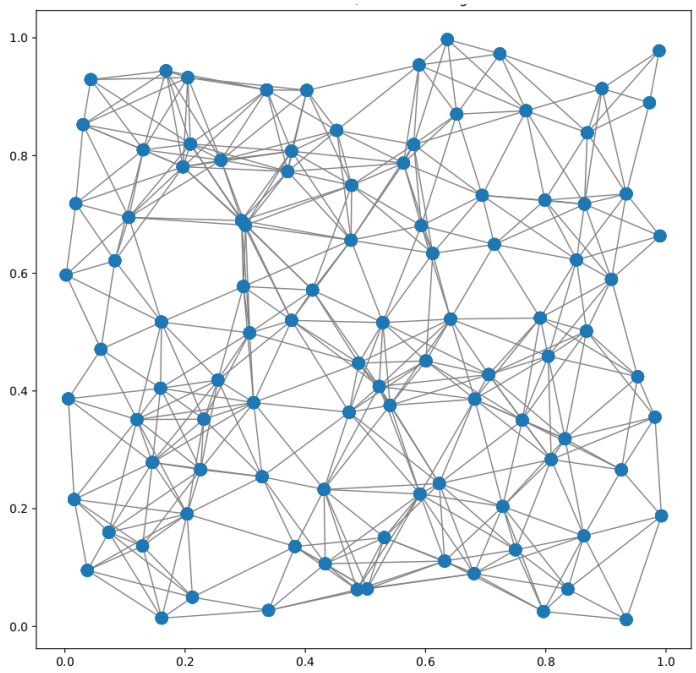
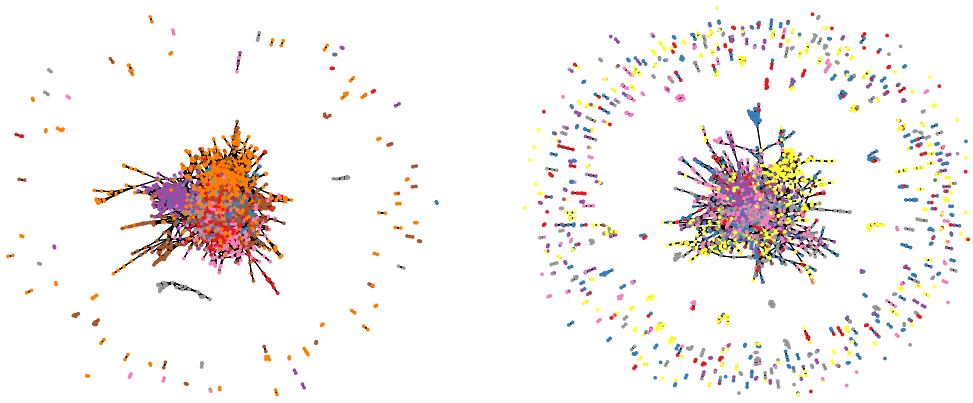


Fig. 2.1 A random sensor graph generated with PyGSP Python package.



(a) Cora (b) Citeseer

Fig. 2.2 Cora and Citeseer citation networks. Each node represents a scientific article. These are colored according to their class membership. There are seven different classes in Cora and six in Citeseer. Two nodes are connected by an edge if one cites the other. These plots were generated with PytorchGeometric library.

Definition 3. *The unnormalized graph Laplacian $L \in \mathbb{R}^{n\times n}$ is defined as $L = D - W$, where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix with elements $D_{ii} = \sum_{j=1}^{n} W_{ij}$.*

Because W is symmetric, so is L, and hence it admits a spectral decomposition of the form $L = \Phi\Lambda\Phi^{\mathsf{T}}$, where the columns of Φ are the n eigenvectors Λ is a diagonal matrix whose entries are the n real eigenvalues $\lambda_1, \ldots, \lambda_n$ associated with the eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$ that are the columns of Φ. The eigenvalues of L define the *spectrum* of the graph and carry information about many of its characteristics.

The following proposition summarizes some important properties of the graph Laplacian [66**?**].

Proposition 1. *The following properties hold for the graph Laplacian L.*

1. *L is symmetric and positive semi-definite,*
2. *L has* 0 *as the smallest eigenvalue, and its corresponding eigenvector is* $\mathbf{1}$.
3. *L has n non-negative eigenvalues* $0 = \lambda_1 \leq \cdots \leq \lambda_n$.
4. *The multiplicity of* λ_1 *corresponds to the number of connected components in the graph.*

2.1.2 Graph Signal Processing

Although graphs are interesting on their own, very often they come with data associated with their nodes. Going back to the citation network example, one could think of the graph as a domain where each node is associated with a document, represented as a feature vector. We are going to say such features are *signals* coming from a graph domain. We formalize this notion in the following definition:

Definition 4. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a graph signal is a function $f : \mathcal{V} \to \mathbb{R}$. Let $n = |\mathcal{V}|$, then f can be represented as a vector in $\mathbb{R}^n$, $(f_1, \ldots, f_n)$.*

Graph Signal Processing (GSP) [60, 6, 55] extends the concepts of classic Signal Processing to data coming from graph domains and provides the necessary tools to perform calculus on discrete structures. This field has served as a framework to formalize a vast variety of problems. For instance, the task of node classification can be addressed with the tools GSP provides once we realize that node labels can be seen as a graph signal (see Figure 2.3).

A central actor in GSP is the graph Laplacian we introduced in Definition 3. An important thing to point out about the graph Laplacian is its nature as an operator that acts upon signals

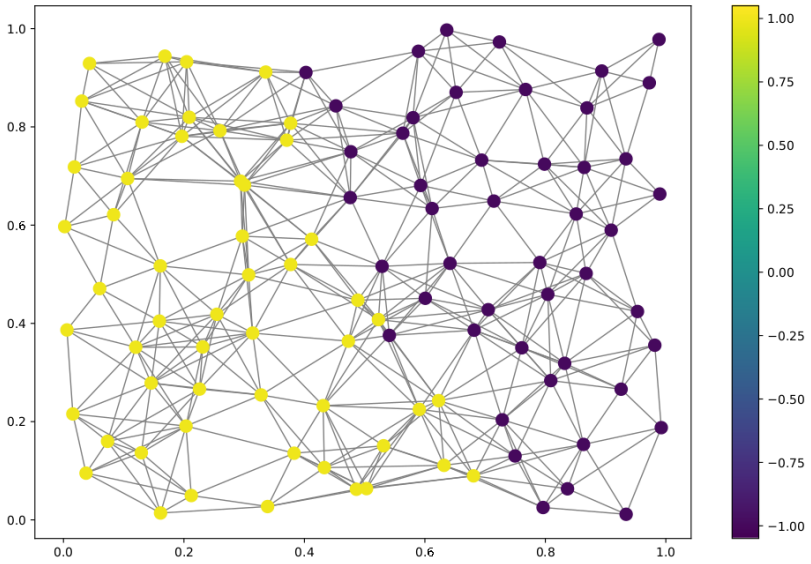


Fig. 2.3 A random sensor graph generated with PyGSP Python package where nodes belong to one of two classes (colored yellow and purple). All the true labels are displayed. In a classification problem many of these labels are missing and we have to infer them.

in the graph measuring how much their steepness changes at each node. To see this, let us first observe that the gradient of a graph signal f boils down to the partial differences between nodes because it realizes in a discrete domain. That is,

$$(\nabla f)_{ij} = f_i - f_j.$$

Now, observe that

$$\begin{aligned}(Lf)_i &= \sum_{j=1}^{n} L_{ij} f_j = \sum_{j=1}^{n} D_{ij} \mathbf{1}_{[j=i]} f_j - W_{ij} f_j \\ &= D_{ii} f_i - \sum_{j=1}^{n} W_{ij} f_j = \sum_{j=1}^{n} W_{ij} f_i - \sum_{j=1}^{n} W_{ij} f_j \\ &= \sum_{j=1}^{n} W_{ij} (f_i - f_j) = \sum_{j=1}^{n} W_{ij} (\nabla f)_{ij}\end{aligned}$$

where $\mathbf{1}_{[.]}$ is the indicator function. In other words, the Laplacian is a difference operator that informs about the local variation of a graph signal.

Another important quantity associated with the Laplacian is the *Laplacian quadratic form* that accounts for the total variability in the graph [66]:

$$
\begin{aligned}
f^{\mathsf{T}} L f &= f^{\mathsf{T}} D f - f^{\mathsf{T}} W f \\
&= \sum_{i=1}^{n} D_{ii} f_i^2 - \sum_{i,j=1}^{n} W_{ij} f_i f_j \\
&= \frac{1}{2}\left[\sum_{i=1}^{n} D_{ii} f_i^2 - 2 \sum_{i,j=1}^{n} W_{ij} f_i f_j + \sum_{i=1}^{n} D_{ii} f_i^2 \right] \\
&= \frac{1}{2} \sum_{i,j=1}^{n} W_{ij} (f_i - f_j)^2
\end{aligned}
\tag{2.1}
$$

We will say the graph signal f is *smooth* with respect to the topology of the graph $\mathcal{G}$ when the quantity of Equation (2.1) is small. To be precise, this quantifies the global smoothness of the graph signal with respect to the graph. Global smoothness has been widely used as a criterion to decide whether the estimation of a partially observed signal fits a graph. As such, it has been incorporated in numerous formulations as a regularization term that penalizes solutions for which Equation (2.1) is large. We will visit some examples in Section 2.2.

The spectrum of L has an interpretation in terms of frequencies: small eigenvalues are associated with low frequencies, and large eigenvalues with high frequencies. As a matter of fact we know that the i-th eigenvalue of L is such that

$$
\begin{aligned}
\lambda_i = \min_{x} \quad & x^{\mathsf{T}} L x \\
\text{s.t.} \quad & x \perp \mathbf{u}_0, \ldots, \mathbf{u}_{i-1} \\
& \|x\| = 1
\end{aligned}
\tag{2.2}
$$

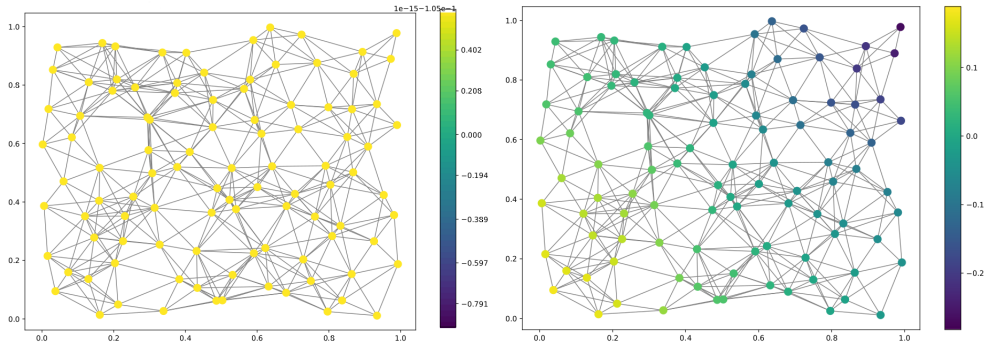
and that the i^{th} eigenvector $\mathbf{u}_i$ is the vector that minimizes that objective. In other words, the eigenvectors minimize the Laplacian quadratic form subject to the restrictions of Equation (2.2), providing a basis to span graph signals in the frequency domain. The first eigenvector is constantly $\mathbf{1}$ as stated in Proposition 1, the second eigenvalue corresponds to the lowest, or smoothest frequency, and so on. This can be visualized in Figure 2.4.

Analogous to Signal Processing, we can decompose a graph signal in terms of the discrete frequencies $\lambda_1, \ldots, \lambda_n$, as stated in the following definition.

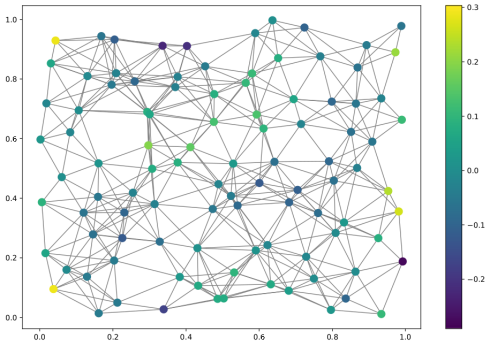
Definition 5. *The Graph Fourier Transform (GFT) $\widehat{f}$ of a graph signal f is*

$$
\widehat{f}(\lambda_i) = \langle f, \mathbf{u}_i \rangle.
$$

The GFT is an important tool that extends certain notions of calculus to non-Euclidean domains. In particular, it permits to generalize the notion of *convolution* to non-Euclidean



(a) Eigenvector corresponding to λ_1 (b) Eigenvector corresponding to λ_2



(c) Eigenvector corresponding to λ_{21}

Fig. 2.4 Plot of eigenvectors corresponding to (a) the first eigenvalue $\lambda_1 = 0$, (b) the second eigenvalue λ_2, and (c) the 21^{st} eigenvalue, λ_{21}. We can see how eigenvectors associated with bigger eigenvalues tend to oscillate more.

domains. Let us take a "detour" to recall some important concepts. The convolution operation acts on two functions (in Euclidean domain) as follows:

Definition 6. *Given two functions h and g, the convolution is defined as*

$$(h * g)(x) = \int h(\tau)g(x-\tau)\partial\tau. \tag{2.3}$$

Let us observe that it is not straightforward to use the convolution defined in Equation (2.3) in the context of graph domains. A workaround is to use the *Convolutional Theorem* to rewrite the convolution operation in terms of the graph spectral domain [6].

Theorem 1. *For functions* $f, g : \mathcal{V} \to \mathbb{R}$ *it holds that*

$$(f * g)(x) = \sum_{i \geq 1} \langle f, \mathbf{u}_i \rangle \langle g, \mathbf{u}_i \rangle \mathbf{u}_i(x). \tag{2.4}$$

In graphs the summation in (2.4) becomes finite, and can be expressed in matrix form as

$$\mathbf{Gf} = \Phi \text{diag}(\widehat{g}_1, \ldots, \widehat{g}_n)\Phi^{\mathsf{T}}\mathbf{f}, \tag{2.5}$$

where $\widehat{g}_i = \widehat{g}(\lambda_i)$, and Φ is the matrix whose columns are the eigenvectors of L. Put in simple words, the recipe for convolving a signal in a graph domain is to get its GFT, apply the bank of filters $\widehat{g}_i$, and then compute the inverse GFT.

Note that in this context, a filter is a function defined in the frequency domain of the graph, $\widehat{g}(\Lambda)$. We can think of filters as functions that "manipulate" the spectrum of the graph, maybe to attenuate some frequencies and to strengthen some others.

As a way of example, let us consider the heat diffusion filter defined as

$$H(\tau) = \Phi e^{-\tau\Lambda}\Phi^{\mathsf{T}}.$$

Figure 2.5 shows the sensor graph defined above after applying the heat kernel for different values of τ.

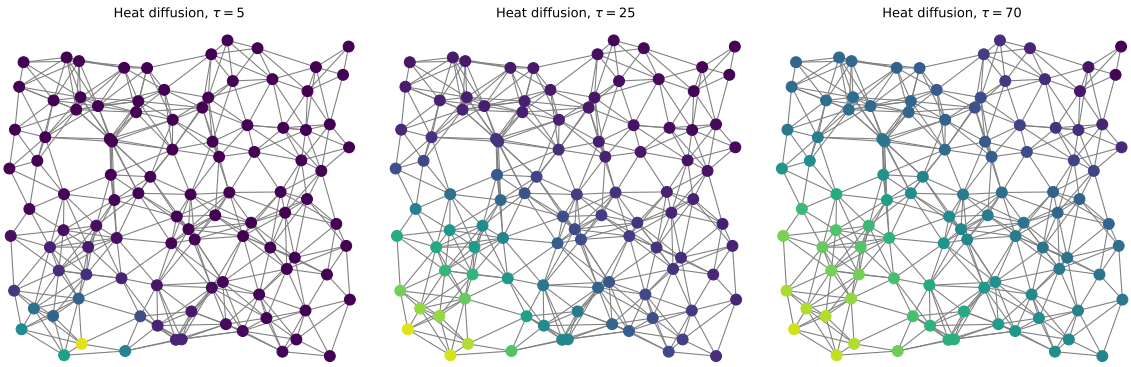


Fig. 2.5 Heat diffusion filter applied to the sensor graph for $\tau = 5$ (left), $\tau = 25$ (center) and $\tau = 70$ (right).

The convolution operation is the centerpiece of the *Convolutional Neural Network* (CNN), a powerful neural network model that is able to encode specific symmetries and constraints. The tools described above allow to extend ideas from CNNs to graphs, leading to the *Graph Convolutional Network* (GCN) model [7, 39].

In Section 2.2.2 we will discuss the role of GCNs in Graph-based Semi-supervised Learning.

2.2 Graph-based Semi-Supervised Learning

Semi-supervised Learning (SSL) is a learning paradigm that exploits both labeled and unlabeled data. SSL algorithms have been successfully applied to problems where labeled data is very scarce, and where there is access to relatively large amount of unlabeled data.

Formally, we assume we have access to a dataset of the form $\mathcal{D} = \mathcal{D}_{\text{sup}} \cup \mathcal{D}_{\text{unsup}}$ where $\mathcal{D}_{\text{sup}} = \{(x_i, y_i)\}_{i=1}^{k}$ is the supervised set and $\mathcal{D}_{\text{unsup}} = \{x_i\}_{i=k+1}^{n}$ is the unlabeled set. We assume each observation (x_i, y_i) comes from a distribution $p(x, y)$ where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. We also assume the supervised set is relatively small with respect to the unlabeled set. The goal is to find a mapping $g : \mathcal{X} \to \mathcal{Y}$ that fits the observed data as best as possible. SSL algorithms work by making some assumptions about the distribution of the input data so that one can leverage the information provided by the unlabeled set. A few popular assumptions are described below.

Smoothness assumption The smoothness or continuity assumption establishes that points that lay in a dense region have outputs that are likely to be close.

Cluster assumption The cluster assumption is a discrete version of the smoothness assumption: points form clusters, and if two elements belong to the same cluster then they are likely to belong to the same class. For a classification task this implies that an optimal decision boundary will pass through a low density region.

Manifold assumption High-dimensional data lies in a low-dimensional manifold. This assumption is key to avoid the "curse of dimensionality".

SSL methods can be classified into *inductive* and *transductive*. The former aims at learning a classifier that is able to handle data from the whole domain $\mathcal{X}$. The latter is only concerned with finding the labels corresponding to the unlabeled elements of the training set.

Graph-based semi-supervised learning (GSSL) algorithms are a class of semi-supervised methods that additionally rely on a graph structure associated with the data. In the taxonomy describe above they are usually placed among the transductive methods. These methods received a lot of attention in the last two decades for many reasons [61]. In the first place, in many applications the data is naturally endowed with a graph. Such a graph is assumed to represent the underlying topology of the data, and it thus provides grounds to develop algorithms that comply with the three assumptions presented above. As a matter of fact, the massive use of Internet results in graph-structured datasets to be more widespread. In the second place, many problems can be formulated as convex programs in a straightforward manner [61]. Finally, graphs are expressive objects and can encode rich information about the data.

At a high-level, GSSL algorithms can be classified into methods that introduce an explicit form of regularization to the objective based on the topology of the data, and more modern methods based on graph neural networks. We will describe these different approaches below. In what follows we will assume the data is endowed with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$ with associated adjacency matrix $W \in \mathbb{R}^{n \times n}$.

2.2.1 Algorithms based on Manifold Regularization

Algorithms under this category are characterized by an objective function that consists of a supervised term and a graph regularization term [61]:

$$l(\widehat{y}) = \sum_{(x_i, y_i) \in \mathcal{D}_{\text{sup}}} l_{\text{sup}}(\widehat{y}(x_i), y_i) + \sum_{x_i \in \mathcal{D}_{unsup}} l_{\text{reg}}(\widehat{y}(x_i)). \tag{2.6}$$

The first term is a regular supervised loss, while the second term ensures the solution will be consistent with the topology of the data. A widely used criterion of graph regularization is that of smoothness: the solution has to be as smooth as possible with respect to the graph in the sense of Equation (2.1).

Some methods use the graph as a propagation operator that propagates labels according to how elements are connected. Examples of these are Label Propagation [75] and Label Spreading [72]. Considering binary labels for simplicity, Label Propagation takes the random walk normalized Laplacian $L_{\text{rw}} = D^{-1}W$, initializes $\widehat{y}^{(0)} = [y_1, \ldots, y_l, 0, \ldots, 0]$ and iteratively updates $\widehat{y}$ as:

$$\begin{aligned} \widehat{y}^{(t+1)} &= L_{\text{rw}} \widehat{y}^{(t)} \\ \widehat{y}_i^{(t+1)} &= y_i \text{ for } i = 1, \ldots, l. \end{aligned} \tag{2.7}$$

When normalized as such, L_{rw} can be deemed a random walk matrix, that is, a matrix whose entries represent transition probabilities. Keeping that in mind, what Equation 2.7 does is to iteratively visit the neighborhood of each node, and diffuse the labels according to the strength of the edges.

Label Spreading differs mainly in two things. First it uses the symmetric normalized Laplacian as propagation operator, that is, it uses $L_{\text{sym}} = D^{-1/2}WD^{-1/2}$, and it allows changes in the predicted labels corresponding to the training set:

$$\widehat{y}_i^{(t+1)} = \alpha L_{\text{sym}} \widehat{y}^{(t)} + (1 - \alpha)\widehat{y}^{(0)},$$

where $\alpha \in [0,1)$ is an hyperparameter controlling to what extent the newly computed label estimation differs from its initialization.

We briefly mention here other approaches that address the same problem. Following a different route, Joachims [32] proposed a cost criterion based on spectral clustering. He formulated the problem as a constrained relaxation of the normalized min-cut of the graph, and optimized a cost consisting of a smoothness term, and a supervised term that keeps the solution close to the initial labeling.

At the same time, Zhu et al. [76] developed a method that share some similarities. The authors choose a real-valued function as a relaxation of the hard labels, and minimize a quadratic energy function that depends on the weight matrix associated with the graph.

Belkin et al. [4] extend the Tikhonov regularization with a smoothness term that penalizes functions that oscillate with respect to the data manifold.

Methods based on random walks [63] define transition probabilities in proportion with the edge weights given by W. To label a point x_i, we start from x_i and transition to other nodes for t steps. We then compute the probability of having started from a point with a label y_i and decide on the label accordingly.

2.2.2 Algorithms based on Graph Neural Networks

In the more recent literature the focus is on methods that use the Graph Neural Network (GNN) architecture [56]. GNNs are neural network models that exploit a given graph structure. In a broad sense they compute node features by iteratively aggregating neighborhood information.

A CNN can be deemed a particular case of a GNN where the data sits on a grid type of graph. Given a n dimensional input f, a this model applies a series of convolutional layers followed by a non linearity, that is, it produces an output $\gamma(x)$ of the form

$$\gamma_l(x) = \sigma\left[\sum_{i=1}^{n}(f_i * g_{l,i})(x)\right]$$

where $g_1,\dots,g_L$ are learnable *filters*. CNNs are well known for having been used in groundbreaking work in the field of computer vision [23, 43, 24]. Its success is due to the fact that they capture interesting spatial and temporal dependencies, rendering their output very good *feature maps*.

As mentioned in Section 2.1.2 Graph Signal Processing provides the necessary tools to generalize convolutional layers to any type of graph domain (not only grids). The Graph Convolutional Network is a type of GNN, and it was first introduced by Kipf and Welling [39] in the context of Semi-supervised classification. Since the computation of filters involves

calculating the potentially costly spectral decomposition of the graph Laplacian (as explained in section 2.1.2), the authors exploited the fact that a graph filter can be approximated by

$$g_\theta(\Lambda) \approx \sum_{i=0}^{k} \theta_i T_i(\widehat{\Lambda})$$

where T_i are (orthogonal) Chebyshev polynomials recursively defined as follows:

$$\begin{aligned} T_0(\lambda) &= 1 \\ T_1(\lambda) &= \lambda \\ T_j(\lambda) &= 2\lambda T_{j-1}(\lambda) - T_{j-2}(\lambda) \end{aligned}$$

and where $\widehat{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_n$, $\lambda_{\max}$ the largest eigenvalue of L. Taking $j = 2$ and applying further simplifying assumptions they propose to approximate the graph convolution of Equation (2.5) as

$$\mathbf{Gf} \approx \theta \widetilde{D}^{-1/2}\widetilde{W}\widetilde{D}^{-1/2}\mathbf{f}, \tag{2.8}$$

where $\widetilde{W} = W + I$ and $\widetilde{D}_{ii} = \sum_j \widetilde{W}_{ij}$ are the "reparameterized" adjacency and degree matrices. The GCN is considered a non-spectral model given that it avoids having to calculate the eigenvalues and eigenvectors of the Laplacian matrix.

To perform semi-supervised learning the authors used the formulation in (2.8) to construct a two-layered model of the form:

$$\widehat{y} = \text{Softmax}\big(\widehat{W}\text{ReLU}(\widehat{W}X\Theta_1)\Theta_2\big) \tag{2.9}$$

where $\widehat{y}$ is the estimation of the true labeling y, and $\widehat{W} = \widetilde{D}^{-1/2}\widetilde{W}\widetilde{D}^{-1/2}$. The model of Equation 2.9 applies two layers of convolution, which means the model hops twice through the neighborhood of each node. The ReLU activation that acts on the output of the first layer prevents the model from collapsing (this is a standard practice in Deep Learning). The Softmax function will normalize the output of the second layer into a categorical probability distribution.

GCNs produced a performance leap over more traditional methods based on manifold regularization, and are as of today a building block in state-of-the art models [11, 21, 71].

2.2.3 Graph Construction

When the graph is not readily available we need to construct one in order to use a GSSL method. Graph construction techniques can be classified into those that are task independent or task dependent [62].

Among the task independent graph construction methods the most popular are:

kNN graph We connect each node with its k nearest neighbors in Euclidean distance. Because this graph is not symmetric one may want to further process this graph to keep edges that connect nodes such that one is among the first k nearest neighbors of the other and vice versa.

ε graphs An edge connects two nodes if their distance is smaller than a threshold ε.

Gaussian kernel graph This is a fully connected graph where an edge is assigned a weight proportional to the similarity between two nodes as: $W_{ij} = \exp(-\left\|x_i - x_j\right\|^2 / (2\sigma^2))$, where σ is a parameter that controls the variance of the neighborhoods.

Note that although all these methods follow different criteria they have something in common: edges encode a notion of "similarity" between points. This results in graphs that carry information about the neighborhoods of nodes. Intuitively, if we know something about a node, we can extrapolate that knowledge to neighboring nodes. This "locality" of the graph is closely related to the assumptions described above in Section 2.2.

However, using these heuristics on the original data involves making a strong assumption. That is, assuming that the Euclidean distance does a good job at capturing all the information we need about the relationships between features. Since this is often not the case one may want to turn to task dependent construction methods. In the simplest case, one can use an unsupervised graph learning method. Methods of this kind do not learn a transformation of the data, but rather aim at finding a structure that fits the original data. Alternatively, one could learn a transformation of the data such that heuristic graphs yield better results. One of the first attempts along these lines is the algorithm proposed by [3], which consists in training a supervised classifier on labeled points and using the soft label predictions as the representation to build the graph. If it is possible to construct a set of restrictions that inform about similarities and dissimilarities of data points explicitly, one can use a Metric Learning algorithm to learn an appropriate distance that generalizes to all the dataset. Metric Learning algorithms can be framed in this category because they have a Representation Learning interpretation. We will introduce Metric Learning and discuss some of its most popular algorithms in the next section.

2.3 Metric Learning

We humans (and some animals), have the capability of judging whether two events or objects are "similar" [65, 28]. This notion helps to make decisions based on our experience, the rationale being that, if a phenomenon produces a similar outcome to what we have previously observed, we may respond in a similar manner. Metric Learning [5] is a field of Machine Learning that aims at emulating this behavior mathematically, with the hope of incorporating "similarity/distance judgments" into the automation of processes.

To attain that goal we first need to formalize the notion of "similarity" and "distance". Roughly speaking, provided elements in an arbitrary set, we can measure how similar they are through a *metric*.

Definition 7. *Let $\mathcal{X}$ be an arbitrary set. A metric is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ such that:*

(i) $d(x,y) = d(x,y)$ *for all* $x, y \in \mathcal{X}$,

(ii) $d(x,y) \leq d(x,z) + d(z,y)$,

(iii) $d(x,y) = 0$ *if and only if* $x = y$.

An example is the Euclidean distance defined over $\mathbb{R}^n$ as $d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$.

Note that many machine learning methods that rely on some notion of similarity, like nearest-neighbor classification, clustering, and kernel methods, use the Euclidean distance by default despite the fact that this choice may be a poor one. For example, let us consider problems for which specific features are more important that others. Then, this hierarchy of feature "relevance" is not reflected in the chosen distance. For a task of interest, if one can collect information about which elements are similar and which are dissimilar, then a Metric Learning algorithm can fit an appropriate metric in a data-driven manner.

A typical input to a Metric Learning algorithm comes in the form of a dataset $X \subseteq \mathcal{X}$ associated with a set of restrictions that indicate which elements in X are more similar. For example, restrictions on X can have the form of two sets, namely, a set of similar pairs

$$\mathcal{S} = \{(x^{(i)}, x^{(j)}) : x^{(i)} \text{ is similar to } x^{(j)}\}$$

and dissimilar pairs

$$\mathcal{D} = \{(x^{(i)}, x^{(k)}) : x^{(i)} \text{ is dissimilar to } x^{(k)}\},$$

or a set of triplets

$$\mathcal{R}(X) = \{(x^{(i)}, x^{(j)}, x^{(k)}) : x^{(i)} \text{ is more similar to } x^{(j)} \text{ than to } x^{(k)}\}$$

It is often cheaper and more straightforward to construct this kind of restriction set than to annotate data with labels. For example, in a database of images of celebrities where only one image per celebrity is annotated, it is perhaps easier to select a few pictures that correspond to the same celebrity (similar items) and learn a metric that naturally clusters pictures by celebrity, than to individually annotate each picture. This learning paradigm is often referred to as *weakly supervised learning*.

Provided with that information one would then propose a parametric model that represents the metric or similarity function of interest, and find the optimal parameters subject to the constrains imposed by the set of restrictions.

A popular choice of model is that of Mahalanobis distance, which is defined as

$$d_A(x,y) = \sqrt{(x-y)^\mathsf{T} A(x-y)}$$

where A is a symmetric positive semi-definite matrix that parameterizes the metric[1].

Let us note the Representation Learning aspect there is to Metric Learning. Because A is positive semi-definite it admits a decomposition of the form $A = L^\mathsf{T} L$. Hence, one can rewrite equation (2.3) as

$$\begin{aligned} d_A(x,y) &= \sqrt{(x-y)^\mathsf{T} A(x-y)} && (2.10)\\ &= \sqrt{(x-y)^\mathsf{T} L^\mathsf{T} L(x-y)} && (2.11)\\ &= \|Lx - Ly\|_2\,, && (2.12) \end{aligned}$$

which implies that learning a Mahalanobis distance is equivalent to the task of learning a linear representation function Lx such that similar elements are grouped according to the Euclidean distance.

The earliest approach to optimize a Mahalanobis distance was proposed by [68]. Their objective function is defined as

$$\begin{aligned} A^* = \min_{A \in S_+^d} \quad & \sum_{x^{(i)}, x^{(j)} \in \mathcal{D}} d_A(x^{(i)}, x^{(j)}) \\ \text{s.t.} \quad & \sum_{x^{(i)}, x^{(j)} \in \mathcal{S}} d_A(x^{(i)}, x^{(j)}) \leq 1 \end{aligned}$$

where S_+^d is the cone of $d \times d$ symmetric positive semi-definite matrices, $\mathcal{S}$ is a set of similar items, and $\mathcal{D}$ is a set of dissimilar items. The intuition behind this algorithm is that we

[1] To be rigorous, note that this actually defines a *pseudo-metric* because the property (iii) in Definition 7 is true only in the $\Leftarrow$ direction.

want to push dissimilar objects apart as much as possible restricted to keeping similar items close together. Another example of Mahalanobis distance learning is Large Margin Nearest Neighbors [67]. This algorithm is supervised because it assumes access to a training set of the form $\{(x^{(i)}, y^{(i)})\}$ where $y^{(i)}$ is the label of $x^{(i)}$. The main idea is to construct a set of similar pairs and a set of triplets with the following criteria:

$$\mathcal{S} = \{(x^{(i)}, x^{(j)}) : y^{(i)} = y^{(j)} \text{ and } x^{(j)} \text{ is among the } k \text{ nearest neighbors of } x^{(i)}\} \tag{2.13}$$

$$\mathcal{R} = \{(x^{(i)}, x^{(j)}, x^{(k)}) : (x^{(i)}, x^{(j)}) \in \mathcal{S} \text{ and } y^{(i)} \neq y^{(k)}\} \tag{2.14}$$

Then, the following convex program is optimized:

$$\begin{aligned} \min_{A \in \mathbb{S}^d_+, \xi \geq 0} \quad & (1-\mu) \sum_{(x^{(i)}, x^{(j)}) \in \mathcal{S}} d_A^2(x^{(i)}, x^{(j)}) + \mu \sum_{i,j,k} \xi_{ijk} \\ \text{s.t.} \quad & d_A^2(x^{(i)}, x^{(k)}) - d_A^2(x^{(i)}, x^{(j)}) \geq 1 - \xi_{ijk} \quad \forall (x^{(i)}, x^{(j)}, x^{(k)}) \in \mathcal{R} \end{aligned}$$

where μ is a coefficient controlling the trade-off between bringing similar elements close together and pushing dissimilar elements far apart, and where ξ acts as a margin. These are two methods among many others that learn linear metrics [8, 70, 27, 26, 57, 14].

The alternative parameterization presented in Equation (2.10) provides a natural way to learn non-linear metrics. To see this let us note that we can write this equation in its more general form

$$d_\phi(x, y) = \|\phi(x) - \phi(y)\| \tag{2.15}$$

where ϕ is an arbitrary function with domain in $\mathcal{X}$. In the case of the Mahalanobis distance we have $\phi(x) = Lx$. This form permits to learn an Euclidean distance on a potentially non-linear transformation of the data, a method often referred to as the *kernel trick*. That is, using a kernel trick we can implicitly transform the initial data into a space that captures non linear relationships between the elements, and fit a linear metric on that transformed space. Some examples are the methods proposed by Davis et al. [14], Hoi et al. [29].

Recent works aim at learning a non-linear metric explicitly by parameterizing the function ϕ with a neural network. The seminal work of Chopra et al. [12] proposed a method along this line. A more recent example is the work done by [31].

2.4 Latent Variable Models and Variational Inference

When studying a phenomenon, the practitioner will usually observe a set of features or *variables* that produce a certain response. However it is often the case that the proposed

model is partial or incomplete in the sense that the observed response is also affected by a set of *unobserved* or *latent variables*. A real-world example from the field of sociology is racial prejudice: one sometimes cannot explicitly measure such attribute, but one can infer it from other pieces of information such as political stances or whether the person approves of a specific legislation [19].

Let us denote random variables in bold. In latent variable models we assume a phenomenon that is observed through a set of known variables $\mathbf{X} = [\mathbf{X}_1, \ldots, \mathbf{X}_d]$ can be explained in terms of a set of latent variables $\mathbf{Z} = [\mathbf{Z}_1, \ldots, \mathbf{Z}_m]$. Mathematically, let p_θ be the probability distribution of $\mathbf{X}$ with parameters θ. Then,

$$p_\theta(\mathbf{X}) = \int p_\theta(\mathbf{X}|\mathbf{Z}) p(\mathbf{Z}) \partial \mathbf{Z} \tag{2.16}$$

where $p(\mathbf{Z})$ is some prior over $\mathbf{Z}$. We are interested in learning the parameters of the likelihood $p_\theta(\mathbf{X}|\mathbf{Z})$, often referred to as the *generative model* of $\mathbf{X}$, and we will assume a specific prior over the latent variables. Furthermore, we want to parameterize our models with neural networks, something that quickly renders the inference problem intractable.

Fitting the parameters of the model in (2.16) amounts to maximizing the log-likelihood, that is, $\max_\theta \log p_\theta(\mathbf{X})$. Provided we have access to a dataset $\mathcal{D} = \{X^{(1)}, \ldots, X^{(m)}\}$, we would have to compute the posterior distribution $p_\theta(\mathbf{Z}|\mathbf{X})$. The challenge is that in many cases the true posterior is intractable. In fact, unless we pick very simple models for the prior and likelihood, such as Gaussians or conjugate distributions, the Equation (2.16) will be very hard or impossible to calculate analytically.

A well known method to alleviate this issue is Expectation-Maximization introduced by Dempster et al. [15], an iterative algorithm that finds the maximum likelihood solution of Equation (2.16). This algorithm alternates between an "E step" and an "M step" until convergence. In the E step we fix the parameters θ and estimate the expected value of the latent variables. In the M step we maximize the expectation function obtained in the E step with respect to θ.

Another widely used method is Variational Inference (VI), proposed by Jordan et al. [33]. Roughly speaking, VI approximates the true posterior $p_\theta(\mathbf{Z}|\mathbf{X})$ with a distribution $q_\phi(\mathbf{Z})$ by maximizing the *evidence lower bound* (ELBO), a quantity that bounds the log-likelihood of

the data as follows:

$$\begin{aligned}
\log p_\theta(\mathbf{X}) &\geq \mathbb{E}_{q_\theta(\mathbf{Z})}\left[\log \frac{p_\theta(\mathbf{X},\mathbf{Z})}{q_\phi(\mathbf{Z})}\right] \\
&= \mathbb{E}_{q_\theta(\mathbf{Z})}\left[\log \frac{p_\theta(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})}{q_\phi(\mathbf{Z})}\right] \\
&= \mathbb{E}_{q_\theta(\mathbf{Z})}\big[\log p_\theta(\mathbf{X}|\mathbf{Z})\big] - \mathbb{E}_{q_\theta(\mathbf{Z})}\left[\log \frac{p(\mathbf{Z})}{q_\phi(\mathbf{Z})}\right] \\
&= \mathbb{E}_{q_\theta(\mathbf{Z})}\big[\log p_\theta(\mathbf{X}|\mathbf{Z})\big] - \mathtt{KL}(q_\phi(\mathbf{Z})||p(\mathbf{Z})) \\
&= \text{ELBO}(\theta,\phi),
\end{aligned} \tag{2.17}$$

where $\mathtt{KL}(q||p)$ is the Kullback–Leibler divergence between distributions defined as:

$$\mathtt{KL}(q||p) = \int p(x)\log\frac{p(x)}{q(x)}\partial x$$

Variational Inference can be seen as the Bayesian generalization of Expectation-Maximization where the parameters of the distribution, θ, are random variables that are not constrained to being a point mass, and that can then be collapsed into the vector of latent variables. The uncertainty over θ is then accounted for in the predictive distribution. This flexibility we get from being able to choose the distribution of the parameters can alleviate the issue of intractability we may encounter in the EM algorithm.

More recently, Kingma and Welling [37] propose the Auto-Encoding Variational Bayes (AEVB) method, an algorithm to scale Variational Inference to large datasets and arbitrary complicated choices of model, like models parameterized by deep neural networks. The authors introduce a recognition model $q_\phi(\mathbf{Z}|\mathbf{X})$, normally parameterized by a Neural Network, and they propose to approximate the expectation $\mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X})}\big[\log p_\theta(\mathbf{X}|\mathbf{Z})\big]$ of Equation (2.17) with Monte Carlo samples. The goal of this method is to maximize the quantity of Equation (2.17). In other words, the goal is to optimize the following loss for the i^{th} sample:

$$\begin{aligned}
\mathcal{L}(\theta,\phi;X^{(i)}) &= -\text{ELBO}(\theta,\phi;X^{(i)}) \\
&= \mathbb{E}_{q_\theta(\mathbf{Z}|X^{(i)})}\big[-\log p_\theta(X^{(i)}|\mathbf{Z})\big] - \mathtt{KL}(q_\phi(\mathbf{Z}|X^{(i)})||p(\mathbf{Z}))
\end{aligned} \tag{2.18}$$

with respect to the model parameters θ and ϕ. Note that it is not straightforward to obtain gradients for that expectation. In fact, the likelihood is a function of the latent variable $\mathbf{Z}$, and for a function f, it is in general difficult to express the gradient of the expectation $\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X})}[f(\mathbf{Z})]$ as the sum of the gradients. This is because we are taking expectation

with respect to a distribution parameterized by ϕ as well. To overcome this issue the authors introduce the *reparameterization trick*, rewriting samples $Z \sim q_\phi(\mathbf{Z}|\mathbf{X})$ as

$$Z = g_\phi(\varepsilon, X)$$

where g_ϕ is a differentiable function and $\varepsilon \sim p(\varepsilon)$, $p(\varepsilon)$ an appropriate distribution. This makes it possible to rewrite the gradients of the expectation for the i^{th} sample as follows:

$$\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X})}[f(Z^{(i)})] \approx \frac{1}{L}\sum_{l=1}^{L} \nabla_\phi f(g_\phi(\varepsilon^{(l)}, X^{(i)}))$$

In other words, the reparameterization trick allows us the express the gradient of the expectation as (approximately) an expectation of the gradient, thus providing an optimization perspective of the inference problem we described. Using this trick and assuming a closed form for the KL divergence we can approximate equation (2.18) as:

$$\mathcal{L}(\theta, \phi; X^{(i)}) \approx \frac{1}{L}\sum_{l=1}^{L} -\log p_\theta(X^{(i)}|Z^{(il)}) + \mathtt{KL}(q_\phi(\mathbf{Z}|X^{(i)})||p(\mathbf{Z}))$$

where $Z^{(il)} = g(\varepsilon^{(il)}, X^{(i)})$ and $\varepsilon^{(il)} \sim p(\varepsilon)$. In general we take $L = 1$ to approximate the likelihood function.

Later on in Chapter 4 we will use the AEVB framework to jointly model a graph and a semi-supervised learning model.

2.5 Unsupervised Graph Learning

The goal of graph learning is to discover the graph structure that better fits a given dataset. That is, given a dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$ of observations from a graph signal with domain in some graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix W we do not know, what is the graph that best explains the data under some criterion?

In the context of GSSL where we also observe a small set of labels associated with some of the data points, we can use a graph learning algorithm to construct a graph that can be later on plugged into a GSSL pipeline. This is a more sophisticated approach than using an heuristic graph based on Euclidean distance where the graph construction step is unsupervised. We discuss some of these approaches below.

In this section we present some algorithms for unsupervised graph learning, i. e., given a dataset $\mathcal{D}$ find an appropriate structure $\mathcal{G}$ under some criterion.

A largely used criterion to decide whether a particular graph fits a dataset or not is that of global smoothness discussed in Section 2.1.2: we say a graph is a good fit for the dataset of interest if the quantity described in Equation (2.1) is small up to specific constraints that prevent the graph from being trivial, such as imposing a non-zero value for the trace of the graph Laplacian or its Frobenius norm. That is, if we stack our dataset $\mathcal{D}$ into a design matrix $X = [x_1^\top, \ldots, x_n^\top]^\top$, then we want to minimize $\text{tr}(XLX^\top)$. The formulations we present here are based on smoothness.

Daitch et al. [13] show that fitting a graph to a dataset $\mathcal{D}$ under the smoothness criterion is equivalent to solving a quadratic program for a set of edges. The authors propose to iteratively solve the quadratic program for a small set of edges until the graph no longer changes.

Kalofolias [34] shows that a smoothness regularization is equivalent to enforcing the solution to be sparse, and uses this fact to write a cost function on W that combines an $L1$ penalization, a log barrier term that enforces sparsity (thus smoothness) and that also prevents the solution from being trivial, and a $L2$ regularization term. He defines a cost of the form

$$\min_{W \in \mathcal{W}} \|W \circ Z\|_1 - \alpha \mathbf{1}^\top \log(W\mathbf{1}) + \frac{\beta}{2} \|W\|_2^2 \tag{2.19}$$

where $\mathcal{W}$ is the set of symmetric matrices with positive entries and zero diagonal, Z is defined to have elements $Z_{ij} = \left\|x_i - x_j\right\|^2$, and α and β are two hyperparameters that allow to control the trade-off between the sparsity of the graph and the strength of the edge weights. Intuitively, the first term is the smoothness loss. We can think of the other two terms as a regularization loss that acts as a trade-off between sparsity and $L2$ norm: the log-barrier is also an entropy regularizer that favors sparse solutions while the squared norm is a shrinking term that prevents the graph from having very large weights. A large β hyperparameter leads to a dense graph, and vice versa.

Dong et al. [18], Lake and Tenenbaum [41] take a different route and focus on the equivalent problem of learning a graph Laplacian. Both methods exploit the fact that a graph Laplacian can be seen as a precision matrix that parameterizes a Gaussian generative model of the features [77]. Imposing a prior on the graph, they derive convex programs from the posterior distribution of the graph.

We note that the problem of learning a graph Laplacian shares some similarities with the problem of learning the covariance matrix of a Gaussian graphical model [22]: as we said above, a graph Laplacian indeed corresponds to the precision matrix of a Gaussian distribution. However these should not be mistaken, as the latter is not concerned about

learning a valid Laplacian matrix (for example, we could learn a precision matrix with positive off-diagonal values).

Chapter 3

Joint Learning of the Graph and the Data Representation

In this chapter we propose a model to jointly learn a data representation and a graph from both labeled and unlabeled data such that (i) the learned representation indirectly encodes the label information injected into the graph, and (ii) the graph provides a smooth topology with respect to the transformed data. Plugging the resulting graph and representation into existing graph-based semi-supervised learning algorithms like label spreading and graph convolutional networks, we show that our approach outperforms standard graph construction methods on both synthetic data and real datasets.

Contents

3.1 Introduction

As discussed in Chapter 1, an important bottleneck for the development of accurate Natural Language Processing (NLP) tools for many applications and languages is the lack of annotated data. We are then interested in graph-based SSL as a means to annotate data in a data-driven manner.

Recall that in graph-based SSL methods the graph is used as a propagation operator to transfer labels from labeled to unlabeled points. Despite differences in the way this

propagation is achieved, graph-based SSL approaches all rely on two assumptions: (i) the graph representing the data provides a faithful approximation of the manifold on which the data actually live, and (ii) the underlying labels are smooth with respect to this manifold.

The challenge is that very often there is no *a priori*-known graph, which raises the question of how to best construct this graph over the dataset given some data representation. In Section 2.2.3 we described methods to build graphs based on heuristics. Even though these choices are straightforward to compute they may poorly adapt to the intrinsic structure of the data manifold and hence violate assumption (i). We can decide to use a more sophisticated graph construction method, as those we presented in Section 2.5, but all these approaches heavily depend on the choice of data representation and disregard the label information, making them unable to adapt to the prediction task and therefore potentially violating assumption (ii). While supervised representation learning techniques such as metric learning (Section 2.3) could be used to adapt the representation to the task of interest, for instance by bringing closer points with the same label, the lack of labeled data in the semi-supervised learning scenario makes them prone to overfitting.

In this chapter we will describe an original semi-supervised algorithm for graph construction that adapts to both the data and the predictive task. Specifically, our approach leverages the labeled and unlabeled data to jointly learn a graph and a data representation. On the one hand, the graph is learned to provide a smooth topology with respect to the learned representation. On the other hand, the representation should bring closer (labeled and unlabeled) points that are neighbors in the graph as well as similarly labeled points, while pulling away points of different labels. A key feature of our approach is that the learned representation indirectly encodes and injects label information into the graph beyond the labeled points alone. We formulate our problem as a joint optimization problem over the representation and the graph weights, with a hyperparameter to easily control the sparsity of the resulting graph and thereby obtain a good approximation of the underlying manifold. We discuss some appropriate parameterizations for learning the representation, which revolve around adapting pre-trained embeddings so as to avoid overfitting. We then propose to solve our joint problem by alternating optimization on the representation and the graph. We validate our approach through several graph-based SSL experiments using label spreading [73] and graph convolutional networks (GCN) [39], both on synthetic and real text classification datasets. Incidentally, note that our approach is generic and could in principle be used in combination with any existing graph-based SSL framework. The results show that our approach outperforms previous methods which rely on heuristic graphs, generally by a considerable margin. Interestingly, we also observe that our approach effectively bridges

the accuracy gap between a simple method like label spreading and a richer neural-based approach like GCN.

The rest of this chapter is organized as follows. We describe our approach and algorithm in Section 3.2, we present our experimental results in Section 3.3. We finally conclude with future work directions in Section 3.4.

3.2 Model

Our approach learns a graph and a data representation for use in downstream graph-based SSL algorithms. In this section, we start by introducing our formulation as a joint optimization problem over the representation and the graph. We then discuss some relevant choices for the parameterization of the learned representation, and finally present our alternating optimization scheme.

Before delving into the method let us recall that we are in a semi-supervised setting like the one we described in Section 2.2. That is, our dataset consists of a set $\mathcal{D}_{\text{sup}} = \{(x_i, y_i)\}_{i=1}^{k}$ of labeled items and a set $\mathcal{D}_{\text{sup}} = \{x_i\}_{i=k+1}^{n}$ of unlabeled items, $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

We are interested in finding a graph in $\mathcal{W}$, the set of all the $n \times n$ symmetric matrices with zero diagonal, and finding the missing labels $y_{k+1}, \ldots, y_n$. It is important to point out at this point that we are in a transductive setting, meaning that we limit ourselves to find the missing labels and not a function that can generalize to unseen data.

3.2.1 Problem Formulation

For the sake of generality, in this section we formulate our problem with respect to a generic representation function $\phi_\Theta : \mathcal{X} \to \mathbb{R}^k$, parameterized by Θ, which represents any data point $x \in \mathcal{X}$ as a k-dimensional vector $\phi_\Theta(x) \in \mathbb{R}^k$. We discuss some relevant choices of representation functions in Section 3.2.2.

We propose to learn a weighted adjacency matrix W^* and a representation function ϕ_{Θ^*} by minimizing a joint objective function f that involves both the labeled and unlabeled data points:

$$W^*, \Theta^* = \underset{W \in \mathcal{W}, \Theta}{\arg\min}\, f(W, \Theta).$$

Once the above optimization problem has been solved, the learned graph W^* (which is based on the learned representation function ϕ_{Θ^*}) and possibly the representation ϕ_{Θ^*} can then be given as input to any graph-based SSL algorithm to obtain predictions for the unlabeled data.

Our objective function $f(W,\Theta)$ decomposes into three terms:

$$f(W,\Theta) = f_1(\Theta) + \alpha[f_2(W) + f_3(W,\Theta)] \tag{3.1}$$

where $f_1(\Theta)$ and $f_2(W)$ are respectively the representation and graph specific terms, while $f_3(W,\Theta)$ is the joint term. Hyperparameter $\alpha \geq 0$ controls the trade-off between the (supervised) representation learning term f_1 and the unsupervised part (f_2 and f_3).

We now define these three terms. For notational convenience, let us denote by $Z \in \mathbb{R}^{n\times n}$ the matrix whose entries are the normalized squared Euclidean distances between data points in the transformed space, i.e.

$$(Z_\Theta)_{ij} = \frac{||\phi_\Theta(x_i) - \phi_\Theta(x_j)||^2}{\sum_{i<j} ||\phi_\Theta(x_i) - \phi_\Theta(x_j)||^2}.$$

The normalization conveniently removes the dependency on the scale of the data and Θ. The representation term $f_1(\Theta)$ is defined on the labeled data points only and takes the following form:

$$f_1(\Theta) = \sum_{\substack{x_i,x_j,x_k\in L \\ y_i=y_j, y_i\neq y_k}} \left[(Z_\Theta)_{ij} - (Z_\Theta)_{ik} + 1\right]_+, \tag{3.2}$$

where $[\cdot]_+ = \max(0,\cdot)$. As discussed in Section 2.3, this is a large-margin triplet loss similar to those used in metric learning [5]: it attempts to learn a representation function ϕ_Θ that brings each point x_i closer to points x_j with the same label than to differently labeled points x_k, with a safety margin of 1. In practice, we can subsample instead of summing over all possible triplets. We use the supervised information of the dataset to construct triplets in a similar manner as in equation (2.13), except that we also force dissimilar elements to be close so as to make it "hard" for the model to identify them as such. That is:

$$\begin{aligned}
S &= \{(x_i,x_j) : y_i = y_j \text{ and } x_j \text{ is among the } k \text{ nearest neighbors of } x_i\} \\
D &= \{(x_i,x_j) : y_i \neq y_j \text{ and } x_j \text{ is among the } k \text{ nearest neighbors of } x_i\} \\
R &= \{(x_i,x_j,x_k) : (x_i,x_j) \in S \text{ and } (x_i,x_k) \in D\}
\end{aligned}$$

The graph term $f_2(W)$ is inspired from the (unsupervised) graph learning approach proposed by Kalofolias [34] discussed in Section 2.5 (see Equation (2.19)):

$$f_2(W) = \beta||W||_F^2 - \mathbf{1}^\top \log(\mathbf{1}^\top W), \tag{3.3}$$

The log-barrier term on the degrees prevents any node from being isolated in the graph, while the Frobenius norm is a shrinkage term over the graph weights. Combined with our joint term (3.5) defined below, hyperparameter $\beta \geq 0$ directly controls the sparsity of the learned graph: the smaller β, the more concentrated the weights of each point on its nearest neighbors in the learned representation (hence the sparser the graph). On the other hand, as $\beta \to +\infty$, the graph becomes complete with uniform weights. Sparsity allows to enforce the locality property (only close points are connected in the graph) which is necessary to obtain a good approximation of the data manifold. It also reduces the computational cost in downstream graph-based SSL algorithms, whose complexity typically depends on the number of edges in the graph.

Other options are possible for $f_2(W)$ depending on the prior we want to have on the structure of the graph. For instance, one may use

$$f_2(W) = (1/\gamma) \sum_{i,j} W_{ij}[\log(W_{ij}) - 1], \tag{3.4}$$

where $\gamma > 0$ is a hyperparameter. This will force the graph to be fully connected.

Finally, we introduce the joint term bringing together the graph and the representation:

$$f_3(W, \Theta) = \mathrm{tr}(W Z_\Theta) = \sum_{i,j} W_{ij} (Z_\Theta)_{ij}. \tag{3.5}$$

This can be seen as a weighted L_1 norm term on W (which is why it induces sparsity), and equivalently written as a quadratic form of the Laplacian matrix of the graph encoded by the symmetric matrix W. It is also used in approaches based on graph Laplacian regularization, but in our case both the graph and the representation are learned in joint manner. This term makes the graph and the representation as smooth as possible with respect to each other on *both labeled and unlabeled points*.

Overall, our joint objective function (3.1) is designed to produce a sparse topology that tends to be smooth with respect to the data manifold and the underlying labeling function through an appropriate representation. We now discuss the choice of representation function ϕ_Θ.

3.2.2 Choices of Representation Functions

Many options are possible for the representation function ϕ_Θ depending on the nature of the data and task at hand. However, it is important to keep in mind that the amount of labeled information is scarce, hence learning complex text representations from scratch is likely to lead to severe overfitting. We argue that it is preferable to adapt pre-trained

representations, which generally requires to optimize much fewer parameters. We give some concrete examples below.

Linear transformation. Pre-trained word embeddings [48, 52] are commonly used to represent texts in a vectorial space, e.g. by averaging the embeddings of the words occurring in a document. In order to adapt the representation to the task, we can learn a simple linear mapping $\phi_\Theta(x) = \Theta x$ which transforms the initial d-dimensional representation into a k-dimensional one, with $\Theta \in \mathbb{R}^{k \times d}$ and $k \leq d$. Such a strategy has been previously explored in the supervised setting to "re-embed" words in a task-specific manner [16]. This is the representation function that we use in our experiments (see Section 3.3).

Weighted combination. Recent work in learning deep contextualized word representations such as ELMo [54] and BERT [17] allows to learn a task-specific combination of the token representations obtained at the K layers of the model, which typically capture different aspects of tokens (from syntax to semantics). In this case, we have K initial d-dimensional representations $x \in \mathbb{R}^{K \times d}$ for each text x and we learn a weighted combination $\phi_\Theta(x) = \Theta x \in \mathbb{R}^d$ where $\Theta \in \mathbb{R}^K$ is simply a K-dimensional parameter vector.

3.2.3 Optimization

We propose to optimize the cost function $f(W, \Theta)$ by alternating minimization over W and Θ, which is guaranteed to converge to a local optimum. This is a natural approach: one step learns a smooth graph given the current representation Θ, while the other learns a smooth representation with respect to the current graph (this can be seen as a regularizer for Θ based on unlabeled data) and also tries to keep labeled points of the same class closer than points of different class.

As the joint problem is nonconvex, initialization plays an important role. We propose to initialize the graph weights to zero and to start by optimizing Θ so that the initial representation focuses only on the (scarce) labeled data. The graph learned on this representation will thus strongly connect together the labeled points as well as unlabeled points that are very close to the labeled points and are thus likely to share the same label. At the next iteration, these unlabeled points will then contribute in learning a better representation and in turn a graph which strongly connects new unlabeled points. This process can be seen as a principled version of self-training heuristics popular in traditional (non-graph-based) semi-supervised learning [64].

The subproblem of optimizing W given Θ is convex regardless of whether we define $f_2(W)$ as (3.3) or (3.4). Using (3.4) is computationally convenient as the subproblem has

a closed-form solution: the weights are exponentially decreasing with the distance in the current representation ϕ_Θ, as given by the radial kernel $W_{ij} = \exp(-\gamma(Z_\Theta)_{ij})$ [34]. Note that unlike the classic radial kernel baseline construction method mentioned in Section 2.2.3, our graph is computed based on the learned representation ϕ_Θ by minimizing the joint objective function with respect to W. One drawback of using (3.4) is that the resulting graphs are always fully connected. Using (3.3) instead, we can obtain sparse graphs but the solution must be computed with an iterative algorithm. We found that the primal-dual algorithm introduced by [34] converges slowly in practice — we instead optimize W by simple gradient descent over the "effective" $n(n-1)/2$ weights, adding a small positive constant inside the log term in (3.3) to make the objective function smooth.

As ϕ_Θ is typically differentiable in Θ (as in the examples outlined in Section 3.2.2), we also solve the subproblem in Θ by (stochastic) gradient descent. Note that this subproblem is generally nonconvex due to the distance difference in $f_1(\Theta)$.

Remark 1. *Updating W requires to optimize over $O(n^2)$ variables, which was manageable for the datasets used in our experiments. To scale to larger datasets, one can restrict the optimization to the weights corresponding to pairs of points that are close enough in the learned representation space*[1] *(other weights are kept to 0). This has a negligible impact on the solution in sparse regimes (small β).*

3.3 Experiments and Results

In this section, we study the practical behavior of our method by comparing the accuracy of downstream graph-based SSL algorithms when the graph (along with the underlying representation) is learned with our approach (**ours**) rather than constructed with the following baseline strategies:

- **radial**: Complete graph with weights $W_{ij} = \exp(-\gamma\|x_i - x_j\|^2)$.
- **knn**: $W_{ij} = 1$ for x_i in the k-neighborhood of x_j (or vice versa), and $W_{ij} = 0$ otherwise.
- **kalo**: Unsupervised graph learning with the method of [34]. This corresponds to our approach when using the graph term (3.3) and keeping the original representation fixed.

In all cases the graph is constructed over the union of labeled (train set) and unlabeled data (validation and test sets). For experiments with our method, the learned representation is a linear transformation of the initial features as explained in Section 3.2.2.

[1] These can be identified in near-linear time using approximate nearest-neighbor techniques [49].

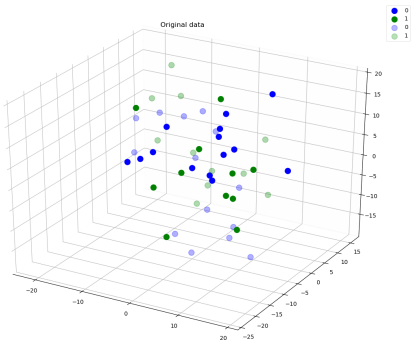


Fig. 3.1 Original 3-dimensional synthetic data. Semi-transparent points are unlabeled.

We perform experiments with two graph-based SSL approaches: Label Spreading (LS) [73], discussed in Section 2.2.1, and the Graph Convolutional Network (GCN) method of [39] discussed in Section 2.2.2. We used the scikit-learn [51] implementation of LS. For GCN, we used the TensorFlow implementation provided by the authors[2] and follow the recommended architecture defined in equation (2.8). We set the number of hidden units h to 16 and λ to 1 as done in [39].

To illustrate the behavior of our approach, we first present some experiments on synthetic data. We then show some results on real text classification datasets.

3.3.1 Synthetic Data

We generated a 3-dimensional dataset consisting of 100 points evenly distributed in two classes (Figure 1). We have two clusters per class placed far from each other while keeping clusters from different classes closer. We randomly picked 60% of the points and removed their labels.

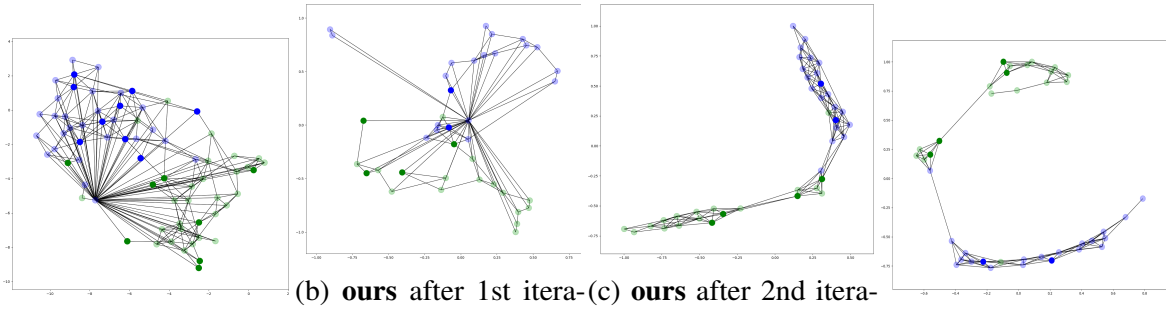
We compare the classification error of GCN and Label Spreading when the input graph is given by our approach instead of using baseline graph construction methods. For GCN, we also give as input the representation learned with our approach. For our approach, we use the graph term (3.3) and for each labeled point x_i, we random sample 2 points x_j of the same class and 3 points x_k of different class and construct all combinations (x_i, x_j, x_k), leading to 6 triplets for each x_i in the triplet loss (3.2). The results given in Table 3.1 show that our approach clearly and consistently outperforms all methods in both GCN and Label Spreading.[3] The improvements are especially large for Label Spreading, as LS makes predictions based on the graph only. In contrast, GCN learns its own (nonlinear)

[2] https://github.com/tkipf/gcn

[3] For this illustrating experiment, we picked the values of hyperparameters giving the best results for each method.

	Label Spreading	GCN
radial	50.7	93.3
knn	81.3	93.3
kalo	77.3	88.0
ours	**96.0**	**96.0**

Table 3.1 Classification accuracy on the synthetic dataset.



(a) **kalo** on original data. (b) **ours** after 1st iteration. (c) **ours** after 2nd iteration. (d) **ours** at last iteration.

Fig. 3.2 Force-directed drawing (spring layout) of graphs learned with **kalo**, and with our method at several iterations of our alternating optimization algorithm. Semi-transparent points are unlabeled.

transformation of the representation given as input in an end-to-end manner. Still, our method is able to provide some gains for GCN as well, by providing it with a better graph. Note for instance the significant improvement compared to **kalo**, which learns the graph on the original representation.

To visualize this difference, Figure 3.2a shows the graph learned by **kalo**. Although the graph is learned to minimize the smoothness criterion with respect to the data, it fails to accurately capture the label distribution due to the limitations of the initial representation. Our alternating optimization approach overcomes this issue by learning a task-specific graph through an appropriate representation. In Figure 3.2b-3.2c-3.2d, we can see how label information is gradually injected at each step: after the first iteration, the graph is already significantly more smooth with respect to the underlying labeling and the graph is also sparser, but some edges between differently labeled points as well as an overly connected point remain. The following iterations further improve the graph quality. This explains the better performance obtained in downstream semi-supervised algorithms.

3.3.2 Real Data

We now evaluate our method on three text classification tasks derived from the 20NewsGroups dataset,[4] a collection of documents categorized into 20 topics, each one of which is partitioned into sub-topics. We chose the topics of *computers* with classes IBM and Mac ($n = 1945$ documents), *religion* with classes atheism and Christian ($n = 1796$), and *sports* with classes baseball and hockey ($n = 1993$).

For all datasets, we represent data points using the average token embedding based on word2vec [48]. These embeddings are of dimension $d = 300$ and were trained on a 100B word corpus of Google news data (vocabulary size is 3M).[5]

We experiment with different proportions of unlabeled points in the training set (90%, 75%, 60% and 40%), while the rest of the data is evenly split into a validation and a test set. As commonly done in semi-supervised learning, we train on the union of the (labeled) training set and the (unlabeled) validation and test sets, select the values of hyperparameters based on the accuracy on the validation set, and report the corresponding accuracy on the test set.

To evaluate our approach we optimize the objective (3.1) as described in Section 3.2.3 with the graph term defined as in (3.3). To compute the representation term of our objective defined in (3.2), we construct triplets as follows: for each pair (x_i, y_i) in the labeled set we obtain the closest points with labels other than y_i ("imposters"), and the closest points with label y_i ("targets"). We picked 8 imposters and 3 targets. We tune the hyperparameters α from $\{0.001, 0.01, 1\}$, β from $\{0.00001, 0.001, 0.1, 1\}$, the dimension k of the learned representation from $\{16, 32, 64\}$, and perform early stopping with respect to the number of alternating steps between learning the graph and learning the representation (up to 10 alternating steps). We also tuned the hyperparameters of each baseline method (γ for **radial**, k for **knn** and β for **kalo**) and the trade-off hyperparameter of Label Spreading. Finally, we computed the McNemar test of significance [47] to compare the performance of our method against the best baseline. Results marked with a dagger symbol † yield a statistically significant test for a significance level of 0.05.

Label Spreading. Table 3.2a reports test classification accuracies obtained on the test set for each configuration of dataset and proportion of unlabeled data. Our approach clearly outperforms all baselines, most of the time by a large margin. Also, McNemar test indicates that we tend to be significantly better than the best baseline in the more challenging settings where labeled data is the most scarce. The results also show that learning the representation

[4] http://qwone.com/~jason/20Newsgroups/
[5] https://code.google.com/archive/p/word2vec/

dataset	%	radial	knn	kalo	ours
comp	90	62.76	61.89	63.63	**67.10**†
comp	75	67.35	67.35	69.87	**74.67**
comp	60	70.92	67.76	72.84	**75.58**
comp	40	76.58	70.99	75.86	**77.48**
rel	90	80.47	79.53	80.59	**83.88**†
rel	75	84.74	85.05	84.66	**85.18**
rel	60	85.74	83.36	87.22	**88.26**
rel	40	84.60	85.77	**86.74**†	85.96
sports	90	84.11	81.78	86.55	**95.66**†
sports	75	89.81	90.87	89.93	**96.02**†
sports	60	92.77	91.43	92.64	**97.19**
sports	40	95.43	93.32	95.08	**97.36**

(a) Label spreading

dataset	%	radial	knn	kalo	ours
comp	90	69.60	65.91	**70.36**†	67.97
comp	75	74.55	67.95	73.71	**75.15**
comp	60	**77.78**	68.86	74.21	76.82
comp	40	**81.08**	67.21	80.72	76.76
rel	90	83.06	82.35	81.53	**83.41**
rel	75	83.49	83.62	83.88	**85.57**†
rel	60	83.36	83.21	**86.92**	86.03
rel	40	**88.30**	82.65	87.33	86.16
sports	90	94.70	92.48	93.33	**95.13**†
sports	75	**96.84**	94.85	95.78	96.25
sports	60	**98.80** †	95.85	97.19	96.92
sports	40	**98.77**	97.01	97.72	97.89

(b) GCN

Table 3.2 Classification accuracies of Label Spreading and GCN for different graph construction methods and proportions of unlabeled data. McNemar test to compare **ours** vs. the best baseline is statistically significant for those results marked with a dagger symbol †.

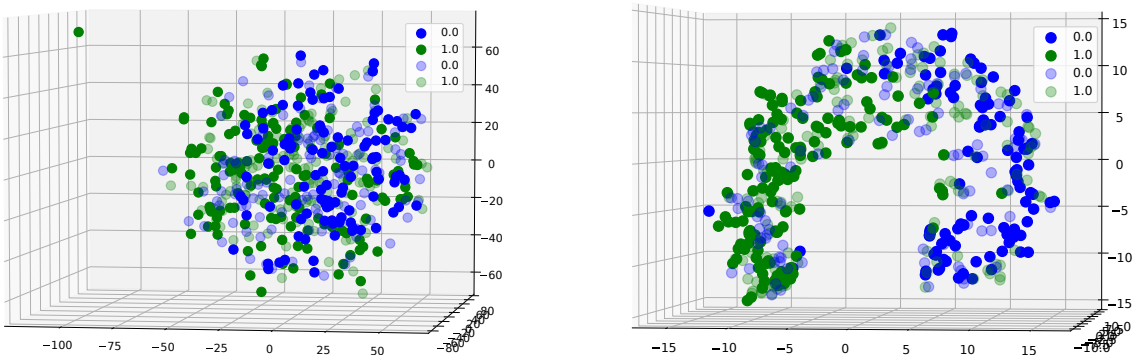
along with the graph makes a clear difference compared to learning the graph only (as seen by the superior performance of **ours** over **kalo**).

As LS only uses the graph to make predictions, these results provide strong evidence of the superior quality of the graphs learned with our method.

Graph Convolutional Networks. We now turn to the more complex GCN prediction model. We re-use the same setup as for LS and feed GCN with both the learned representation and the learned graph.

Table 3.2b summarizes the results. The gains obtained with our approach are smaller than those obtained in LS, which is to be expected since GCN has the ability to learn nonlinear transformations of the data. Nevertheless, we do observe some performance gains, as our approach generally improves upon or closely matches the performance of the best baseline. An interesting finding is that our method tends to close the gap of performance between LS and the richer neural-based GCN model. This suggests that simple propagation approaches may be sufficient in practice for many datasets, if provided with the right graph.

Visualization. We provide visualizations of the representation and the graph learned with our approach on the *rel* dataset. Figure 3.3 shows 3D PCA visualizations of the original representation and the representation learned with our approach. We see that the two



(a) Original representation (b) Learned representation

Fig. 3.3 3D PCA visualization of the original representation (left) and the representation learned with our approach (right) on the *rel* dataset (%75 unlabeled). Transparent dots represent unlabeled documents.

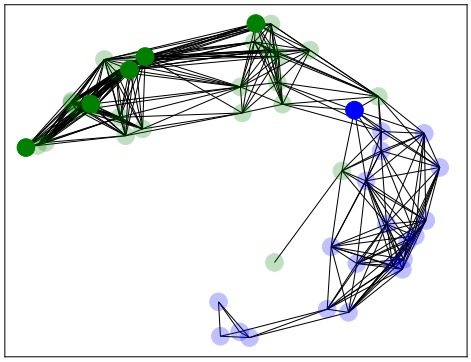


Fig. 3.4 Force-directed drawing (spring layout) of a random 50-node subgraph of the graph learned with our approach on the *rel* dataset (%75 labeled).

classes are quite mixed up in the original representation while the learned representation is much smoother with respect to the underlying labeling (even in this crude low-dimensional summary). Figure 3.4 gives a snapshot of the graph learned with our approach by showing a subgraph of 50 randomly sampled nodes (subsampling helps to avoid clutter). The graph is very smooth with respect to the underlying labeling, and suggests that the learned high-dimensional representation has a nice manifold structure, with some regions of higher densities.

3.4 Conclusion

In this chapter we presented a novel method that brings together graph learning, representation learning and SSL by jointly inferring the graph and the data representation from semi-

supervised data. The output of our approach can then be plugged into any graph-based SSL algorithm instead of using common graph constructions. Our experimental results suggest that the gains are especially significant for graph-based SSL algorithms that are unable to adapt the data representation (like label spreading and its variants), although we observe some gains also for GCN.

It is important to stress the fact that the method here presented is not end-to-end in the sense that we are not fitting a classification model. A path to improving results for richer models such as GCNs could be end-to-end systems that can backpropagate the classification error through the parameters of the classification model (like a GCN) *and* the graph. This is precisely the topic of the next chapter. In what follows we present a end-to-end algorithm where we use a latent variable models framework to account for topology uncertainty.

Chapter 4

Graph Inference and Semi-Supervised Learning with Auto-Encoding Variational Bayes

In this chapter we address the same problem as in Chapter 3, except that we take it a step further: we propose a method to learn a graph structure and the parameters of a semi-supervised model *simultaneously*. We use recent advances in Variational Inference that allow us to model the graph as a latent variable, and to account for the uncertainty in the graph when performing GSSL.

Contents

4.1 Introduction

In the previous chapter we described an algorithm to jointly learn a graph and a data representation in a way that the representation injects label information into the graph. As a result we obtain a task-specific graph and a data representation that can be plugged in a GSSL model. This method can be seen as an intermediate step in a pipeline where we transform the data and learn graph that are better suited for a pre-defined GSSL algorithm.

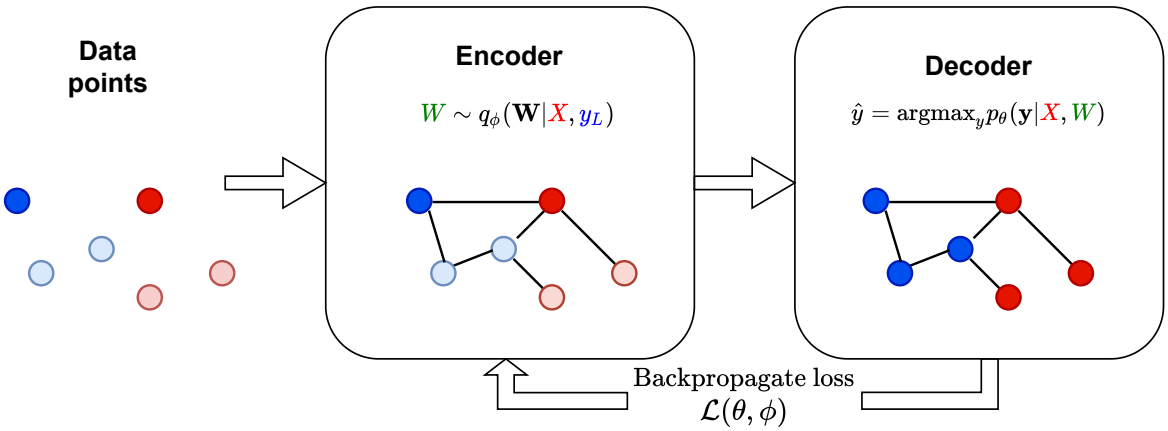


Fig. 4.1 An overview of our probabilistic encoder-decoder framework to jointly learn the graph and the model parameters in graph-based semi-supervised learning.

In this chapter we take this a step forward: we learn the graph and the parameters of a GSSL model simultaneously. We propose a generic probabilistic framework for performing this joint estimation in an end-to-end fashion. Based on Auto-Encoding Variational Bayes, our framework relies on a simple encoder-decoder architecture, as shown in Figure 4.1. First, the encoder takes as input a set of points, along with label information for a subset of them, and outputs a distribution over graphs from which we can sample. Crucially, graph learning is here conditioned on both the input representations and some observed labels. Second, the decoder predicts the unobserved labels from the data points and a graph sampled from the generative model learned in the encoder. There are many possible ways that we could instantiate this framework with specific encoders and decoders. In this chapter we propose a flexible architecture in which the encoder is based on a message passing graph neural network (GNN) [25] that is used to learn the parameters of the graph which define Bernoulli distributions associated with the graph edges. That is, edges are modeled as latent variables, and we learn them by minimizing a reconstruction error over the predicted labels. The decoder, on the other hand, is modeled as a categorical distribution parameterized by a Graph Convolutional Network (GCN) [39]. Crucially, we show that the resulting encoder-decoder model is trainable in an end-to-end fashion: in particular, we are able to back-propagate the classification errors through the parameters of the latent variable graph model. Furthermore, once the parameters of our model have been trained, we can naturally predict the labels of new points unseen at training time.

The probabilistic nature of our framework comes with clear advantages, including the ability to incorporate prior knowledge about the graph or impose specific structures (such as sparsity) upon the resulting topology. We call this method Probabilistic Graph-based

Semi-supervised Learning (PGSSL). This is an important upside over the recent approaches of [9] and [35], which altogether lack a probabilistic interpretation.

Through extensive experiments, we show that our approach delivers significant performance gains over strong baselines, and compares favorably to competitors that require more training data [21].

4.2 Model

We consider a classic semi-supervised learning (SSL) scenario where the learner has access to a dataset where only a small subset of elements are labeled. Formally speaking, we assume the training set has the form $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{l} \cup \{x_i\}_{i=l+1}^{n}$, where $x_1, \ldots, x_n \in \mathbb{R}^d$ are feature vectors and $y_1, \ldots, y_l$ are (discrete or continuous) labels associated with $x_1, \ldots, x_l$, and typically $l \ll n$. We denote by $X \in \mathbb{R}^{n \times d}$ the design matrix formed by the feature vectors, $y_L = [y_1, \ldots, y_l]$ the vector of observed labels and $y_U = [y_{l+1}, \ldots, y_n]$ the vector of missing labels.

Graph-based SSL algorithms rely on a graph whose nodes are the data points and (possibly weighted) edges represent the underlying structure of the data. The graph can be described by a weight matrix $W \in \mathbb{R}^{n \times n}$. In the case of an unweighted graph, $W \in \{0,1\}^{n \times n}$ simply corresponds to the adjacency matrix. Graph-based SSL methods such as Graph Convolutional Networks [39] typically operate in the transductive setting: they seek to infer the missing labels y_U through a predictive model $y = f(x; X, W)$, *assuming that the graph W is observed.*

In this work, we address the problem of performing inference on the missing labels when the graph W is not known. We consider a probabilistic framework as described in Section 2.4 in which the unknown graph is represented by an unobserved latent variable $\mathbf{W}$ (we use bold to denote random variables throughout the paper). In the following, we will denote by $\mathbf{X}$ and $\mathbf{y}$ the random variables associated with the dataset X and labels y. We propose to model $\mathbf{y}$ as generated by a random process that depends on $\mathbf{X}$ and $\mathbf{W}$, which factorizes as

$$p(\mathbf{y}, \mathbf{X}, \mathbf{W}) = p_\theta(\mathbf{y}|\mathbf{X}, \mathbf{W}) p(\mathbf{W}|\mathbf{X}) p(\mathbf{X}), \tag{4.1}$$

where the conditional distribution of $\mathbf{y}|\mathbf{X}, \mathbf{W}$ is parameterized by θ. To perform inference on the missing labels y_U, one would have to calculate the integral

$$p(\mathbf{y}|\mathcal{D}) = \int p_\theta(\mathbf{y}|X, \mathbf{W}) p_\theta(\mathbf{W}|X, y_L) \partial \mathbf{W},$$

which involves the computation of the potentially intractable posterior distribution

$$p_\theta(\mathbf{W}|X, y_L).$$

To alleviate this issue we formalize our problem using the Auto-encoding Variational Bayes (AEVB) framework [37], and introduce the recognition model $q_\phi(\mathbf{W}|\mathbf{X}, \mathbf{y})$ parameterized by ϕ to approximate the true posterior distribution $p_\theta(\mathbf{W}|\mathbf{X}, \mathbf{y})$. In order to learn the parameters $[\theta, \phi]$ of the model, we need to maximize the Evidence Lower Bound (ELBO). The ELBO is a quantity that lower bounds the marginal likelihood $p(y_L|X)$ as:

$$\begin{aligned} \log p(y_L|X) &\geq \mathbb{E}_{q_\phi(\mathbf{W}|X,y_L)}\left[\log \frac{p_\theta(y_L|X,\mathbf{W})}{q_\phi(\mathbf{W}|X,y_L)}\right] \\ &= \text{ELBO}(\theta, \phi; X, y_L). \end{aligned} \tag{4.2}$$

As standard in the AEVB framework, we can decompose (4.2) into a reconstruction error and a KL divergence between the recognition model q and a prior distribution $p(\mathbf{W})$ over the latent graph:

$$\begin{aligned} \text{ELBO}(\theta, \phi; X, y_L) = \; & \mathbb{E}_{q_\phi(\mathbf{W}|X,y_L)}[\log p_\theta(y_L|X,\mathbf{W})] \\ & - \texttt{KL}(q_\phi(\mathbf{W}|X,y_L)||p(\mathbf{W})). \end{aligned}$$

The prior acts as a regularizer to avoid degenerate solutions (e.g., a graph with no edge), but can also be used to incorporate useful background knowledge about the graph. We will get back to the choice of prior in Section 4.2.4.

In the nomenclature of Auto-Encoding Variational Bayes, the approximate posterior q_ϕ is called the *encoder* and the likelihood model p_θ the *decoder*. We can describe the generative process (4.1) in terms of the encoder and decoder as follows: we feed the encoder with the training data $\mathcal{D} = (X, y_L)$ to compute $q_\phi(\mathbf{W}|X, y_L)$, we sample a graph $W \sim q_\phi(\mathbf{W}|X, y_L)$ from the re-parameterized approximation of the recognition model q_ϕ, and then we run the decoder $p_\theta(\mathbf{y}|X, W)$ to obtain $\widehat{y}$, the estimation of the true labeling y. We illustrate this pipeline in Figure 4.1. An interesting consequence of this process is that it is not restricted to the transductive setting, unlike most graph-based SSL approaches. Indeed, once parameters ϕ and θ have been learned, we can naturally predict the labels of new points unseen at training time.

Now that we have introduced our general variational auto-encoder formulation, we can instantiate the different components (encoder/decoder) in various ways depending on the

requirements and background knowledge of the task, the amount of labeled data, etc. The next section presents a flexible architecture that is trainable in an end-to-end fashion.

4.2.1 Architecture

We present an architecture in which we can infer the missing labels and the graph jointly by end-to-end training, i.e., we can backpropagate the label prediction error through the parameters of the latent variable model (encoder). In a nutshell, we instantiate the encoder q_ϕ as Bernoulli distributions over edges parameterized by a Graph Neural Network (GNN), and the decoder p_θ as a categorical distribution parameterized by a Graph Convolutional Network (GCN).

In the following, we describe the encoder and decoder components in more details and discuss possible choices for the prior distribution over the graph. We then present our end-to-end training algorithm.

4.2.2 Encoder

We propose to model $\mathbf{W} = \{\mathbf{W}_{i,j}\}$ as a collection of binary random variables drawn from Bernoulli distributions. In other words, W is an adjacency matrix whose entry (i, j) is drawn as $W_{i,j} \sim \text{Ber}(\pi_{i,j})$, where $\pi_{i,j} \in [0,1]$ represents the probability that an edge connects points x_i and x_j. We denote by $\pi \in [0,1]^{n \times n}$ the matrix of all $\pi_{i,j}$'s. While each edge is drawn independently, we model π jointly as a function of the dataset X parameterized by ϕ:

$$\pi = f_\phi(X), \text{ or equivalently, } W_{i,j} \sim \text{Ber}(f_\phi(X)_{i,j}). \tag{4.3}$$

Note that $\mathbf{W}$ does not depend on $\mathbf{y}$ in this particular setting. We are thus approximating the true posterior $p(\mathbf{W}|X, y_L)$ by an encoder $q_\phi(\mathbf{W}|X) = \prod_{i,j} q_\phi(\mathbf{W}_{i,j}|X)$.

To be able to model complex relationships between points of the dataset, we define $f_\phi(X)$ to be an instance of a message passing Graph Neural Network (GNN) [25]. Roughly speaking, the role of the GNN is to construct edge embeddings and to process them into Bernoulli parameters. The GNN operates over a neighborhood graph which is defined based on distance in the original feature space: for each data point x_i, a set of "neighbors" $\mathcal{N}(i)$ that the GNN will take into account when aggregating edge embeddings.[1]

Formally, the transformation f_ϕ is defined recursively by a series of node and edge embeddings. Denoting by $h_i^{(1)} = f_{\text{node}}^{(1)}(x_i)$ the initial node embedding of each point x_i, the

[1] $\mathcal{N}(i)$ should not be mistaken with the neighborhood of x_i in the graph W involved in the prediction of y_i in the decoder step.

recursion is then given by

$$h_{i,j}^{(k)} = f_{\text{edge}}^{(k)}\big(\omega(h_i^{(k)}, h_j^{(k)})\big) \qquad \forall i \in [n], j \in \mathcal{N}(i) \tag{4.4}$$

$$h_i^{(k+1)} = f_{\text{node}}^{(k+1)}\Big(\sum_{j \in \mathcal{N}(i)} h_{i,j}^{(k)}\Big) \qquad \forall i \in [n]. \tag{4.5}$$

In the equations above, (4.4) produces an edge embedding for all pairs (i, j) such that $x_j \in \mathcal{N}(i)$ from the combination of the current node embeddings of x_i and x_j (ω denotes a fixed combination function, such as concatenation or difference), and (4.5) computes a new node embedding from the aggregation of neighboring edge embeddings.

Finally, after a fixed number of hops K, the edge probabilities are defined as a softmax over the final edge embeddings:

$$\pi_{i,j} = f_\phi(X)_{i,j} = \text{Softmax}(h_{i,j}^{(K)}). \tag{4.6}$$

It is worth noting that the number of hops K defines the extent of the local structure that influences the presence or absence of edges. Choosing a large K leads to a very expressive model which may have large sample complexity and generalize poorly. We note that in a different context (modeling physical systems), Kipf et al. [38] use a similar model with $K = 2$. In our case, supervision comes from the prediction error on the labeled subset of the training data. Keeping in mind that labeled data is often very scarce in semi-supervised learning, in our experiments we keep the encoder simple so as to limit overfitting. In particular, we set $K = 1$ (meaning that we only hop once, i.e. Eq. 4.5 is never executed) and set $\mathcal{N}(i) = [n]$ for all $i \in [n]$. This corresponds to assuming that the probability of observing an edge (i, j) depends only on the features (x_i, x_j) of its two endpoints, which is a common in statistical graph learning [see for instance 50]. We further set the initial node embedding to the identity, i.e., $f_{\text{node}}^{(1)}(x) = x$, let $f_{\text{edge}}^{(1)}$ to be a multi-layer feed-forward neural network and define $\omega(h_i^{(1)}, h_j^{(1)}) = h_i^{(1)} - h_j^{(1)}$ as commonly done in the GNN literature.

4.2.3 Decoder

The purpose of the decoder $p_\theta(\mathbf{y}|X, W)$ is to predict the labels $[y_{l+1}, \ldots, y_{l+u}]$ given a dataset X and a graph W. We choose to instantiate the decoder as a Graph Convolutional Network (GCN) with one hidden layer [39]. Focusing here on a classification task with c discrete labels, this corresponds to modeling the categorical distribution of $\mathbf{y}|X, W$ as:

$$p_\theta(\mathbf{y}|X, W) = \text{Softmax}(\widetilde{W}\,\text{ReLU}(\widetilde{W}\,X\,\theta_1)\,\theta_2), \tag{4.7}$$

where $\theta = [\theta_1, \theta_2]$ are the parameters to be learned, $\theta_1 \in \mathbb{R}^{d \times k}$, $\theta_2 \in \mathbb{R}^{k \times c}$, c is the number of classes, $\widetilde{W} = \widetilde{D}^{-1/2}(W + I)\widetilde{D}^{-1/2}$ is the normalized adjacency matrix, and $\widetilde{D}_{ii} = 1 + \sum_j W_{i,j}$.

4.2.4 Choice of Prior

Given the choice of encoder in Section 4.2.2, the prior distribution decomposes as $p(\mathbf{W}) = \prod_{i,j} p(\mathbf{W}_{i,j})$ where each $\mathbf{W}_{i,j}$ follows a Bernoulli distribution with parameter $\rho_{i,j} \in [0, 1]$. We denote by $\rho = \{\rho_{i,j}\}$ the collection of all prior parameters. Beyond its role as a regularizer, the prior can be used to inject useful knowledge or structure into the model. We give some concrete examples below.

Graph construction heuristics. When no graph is available *a priori*, the classic approach in graph-based SSL is to connect data points using a graph construction method as those described in Section 2.2.3. Based on this idea, we can assign a higher prior probability to edges (i, j) for which $\|x_i - x_j\|$ is small. For instance, building upon the k-nearest neighbor graph heuristics, we can set $\rho_{i,j} = \rho_1$ if x_j belongs to the k-nearest neighbors of x_i and $\rho_{i,j} = \rho_2$ otherwise for some $\rho_1 \geq \rho_2$.

Incomplete/noisy graph. In some applications, a graph is available but may incomplete or noisy (e.g., due to costly or error-prone data collection). We can use the prior to reflect our greater confidence in the presence of some of the edges, for instance by setting $\rho_{i,j} = 1$ for known edges in the case of an incomplete graph. Crucially, by biasing the encoder towards learning embeddings that give large posterior probability to these known edges, our model can naturally discover some missing edges.

Controlling sparsity. The prior can also be used to control the graph sparsity. Indeed, smaller values for $\rho_{i,j}$ will encourage smaller values for $\pi_{i,j}$, and in turn sparser graphs.

4.2.5 End-to-End Training Algorithm

Given the encoder and decoder and a choice of prior over the edges $\rho = p(\mathbf{W})$, our loss function can be written as

$$\begin{aligned}\mathcal{L}(\theta,\phi) &= -\,\mathrm{ELBO}(\theta,\phi;X,y_L)\\ &= \sum_{\substack{i=1\\ W\sim q_\phi(\mathbf{W}|X)}}^{l} -\log p_\theta(y_i|X,W)\\ &\quad + \sum_{(i,j):x_j\in\mathcal{N}(x_i)} \mathtt{KL}(q_\phi(\mathbf{W}_{i,j}|X)||\mathrm{Ber}(\rho_{i,j})),\end{aligned} \tag{4.8}$$

where, with a slight abuse of notation, $\log p_\theta(y_i|X,W)$ denotes the likelihood of label y_i under the model. Observe that the first term can be seen as the reconstruction error we obtain from producing an estimation of the known labels through the encoder-decoder pipeline we described. The second term enforces the encoder to remain as close to the prior as possible. Note that the KL divergence between Bernoulli distribution has a simple closed-form.

We now explain how to train our model in an end-to-end fashion. At each epoch, we start by running the encoder to obtain a distribution $q_\phi(\mathbf{W}_{i,j}|X)$ over all edges (x_i,x_j). The difficulty then is that q_ϕ is a discrete distribution over edges, so we cannot directly backpropagate the error through its samples. To address this issue, we use the concrete distribution [46] to get a continuous approximation of q_ϕ and apply the reparameterization trick to compute the gradients. More specifically, we first draw a vector ξ from a $\mathrm{Gumbel}(0,1)$ distribution and then compute $W_{i,j} = \mathrm{Softmax}((h_{i,j}^{(K)} + \xi)/\tau)$, where τ is a parameter controlling the smoothness of the resulting distribution (the bigger τ, the more it resembles a uniform distribution). Finally, we feed the obtained graph W to the decoder to get a distribution over labels $p_\theta(\mathbf{y}|X,W)$, and backpropagate the loss (4.8) through the decoder and encoder to update the parameters θ and ϕ.

We note that considering all possible edges (x_i,x_j) in each epoch entails a potentially large complexity at training time. A possible way to alleviate this computational cost is by sampling a subset of edges that the GNN will take into account at each epoch. Following a similar approach to [45], we sample a set of "positive edges", that is, edges that have a high prior probability according to the prior distribution, and a set of "negative edges" from a Bernoulli distribution with a small parameter.

Table 4.1 Statistics of the datasets.

DATA SET	# NODES	DIM	$\|y\|$	TRAIN/VAL/TEST
20NEWS3	2756	229	3	20/40/2696
20NEWS4	3952	278	4	50/100/3802
DIGITS	1797	64	10	50/100/1647
CORA	2708	1433	7	140/500/1000
CITESEER	3327	3703	6	120/500/1000

4.3 Experiments and Results

We carried out some experiments to compare our approach against both supervised and semi-supervised baselines on a variety of benchmark datasets and settings.

4.3.1 Datasets

We evaluate the methods on datasets that have been commonly used to benchmark semi-supervised learning in recent work [39, 21, 11]. The main statistics and proportions of labeled/unlabeled data for the datasets are summarized in Table 4.1.

Digits and 20NewsGroups are standard English text classification datasets available in scikit-learn [51] which do not come with any graph information. 20News3 and 20News4 are two subsets of 20NewsGroups corresponding respectively to three topics (i.e., "atheism", "hardware" and "forsale") and four topics (i.e., "cryptography", "medicine", "electronics" and "space"). Note that this is similar to how we constructed the datasets in Chapter 3, except that now we are not focused only on binary classification. The documents are represented by $tf\text{-}idf$ features (excluding stop words and bottom 5% of least frequent words).

Cora and Citeseer [59] are citation network datasets consisting of a set of scientific articles represented as one hot word vector indicating the absence/presence of the corresponding word from a list of unique terms, together with an adjacency matrix where an edge connects two documents if one cites the other.

4.3.2 Competing Approaches and Setup

We compare our method (PGSSL) against supervised and semi-supervised competitors. Supervised baselines include logistic regression (LogReg), linear support vector machines (SVM) and feed-forward neural networks (FFNN). Note that the FFNNs boil down to a GCN model that takes no adjacency matrix as input. Graph-based semi-supervised baselines consist of GCNs fed with a graph computed with classic heuristics. We experiment with a

symmetric k-nn graph (GCN+KNN) where two nodes are connected when one is among the k closest neighbors of the other, and a Gaussian kernel graph (GCN+RBF). In both cases we fix the number of layers of the GCN component to 2, the dropout to 0.5, and we tune the hidden dimension from $\{8, 16, 64\}$. Finally, we also compare our approach to the more competitive model proposed by Franceschi et al. [21] (kNN-LDS), see Section 4.4 for a detailed description.[2]

In general, we follow the same experimental setting as proposed by Kipf and Welling [39]. For kNN-LDS we use the experimental setting as in the paper [21], which requires an extra set of training data. That is, we split the citation network into a training, validation and test set as by Kipf and Welling [39], and further split the validation set into two equal parts, a "validation set" and an "early stop" set. In the context of kNN-LDS, the "validation set" is used to supervise the graph loss. This is the only method that requires an extra set of labeled data.

The main hyperparameters of all approaches are fine-tuned on the validation set. In the case of supervised baselines, we tune the regularization parameter C from $\{0.1, 1, 10, 100\}$, and the parameter γ required for SVM from $\{0.01, 0.1, 1, 10\}$. For the semi-supervised baselines requiring the k-nn graph, we tune k from $\{2, 3, \ldots, 20\}$, and for those that use the Gaussian Kernel graph we tune γ from $\{0.001, 1, 10\}$. The hyper-parameters of kNN-LDS were chosen with the same strategy as in the original paper Franceschi et al. [21], which is based on a 2-way split of the validation set.

For our method, we use the simple architecture described in Section 4.2.1. In particular, the encoder is based on a GNN with $K = 1$ hops. For all datasets, we define the node embedding to be $f^{(1)}_{\text{node}}(x) = x$. The edge embedding $f^{(1)}_{\text{edge}}$ is defined as a fully connected feed-forward neural network with number of layers and hidden dimensions to be tuned as hyperparameters (see below). Finally, the decoder consists of a two-layer GCN with hidden dimension tuned from $\{16, 64\}$ and dropout fixed to 0.2.

For training, we optimize the objective (4.8) using Adam optimizer [36] with a dropout rate of 0.5 for both the encoder and decoder, and we set $\tau = 1$ for the Gumbel sample approximation. To perform early stopping we monitor the reconstruction loss in the held out validation set. Regarding hyperparameter tuning, we performed a grid search over the encoder learning rate in $\{0.001, 0.005\}$ and the decoder learning rate in $\{0.001, 0.01\}$. We fixed the patience of the encoder and decoder to 200, and the decrease rate on the learning rates to 0.5. Preliminary experiments indicated that less parsimonious models tend to yield better results on datasets with lower dimensional feature spaces. We then chose different grids of hyperparameters depending on the dimensionality of the data to tune the architecture

[2] We use the code of the authors available at https://github.com/lucfra/LDS-GNN

of the encoder. For 20news3, 20news4 and digits, we tuned the number of hidden layers from {2, 3} and the hidden dimension of the first hidden layer from {128, 64, 32}. For the citation networks, we searched the number of layers in {3, 5} and the hidden dimension of the first hidden layer in {512, 264, 128}. In all cases, the dimension of the following hidden layers decreases by a factor of 0.5. Finally, when the graph is completely missing, we define the prior using a symmetric k-NN graph with a number of neighbors in $\{10, 20\}$: we set $\rho_{ij} = 0.999$ if (x_i, x_j) is connected in the symmetric k-nn graph and $\rho_{ij} = 0.001$ otherwise.

4.3.3 Results in Transductive Setting

Table 4.2 Classification accuracy means and standard deviations on the held-out set X_{test} for 5 random seeds. We used the paired t-test with a significance level of $\alpha = 0.05$ to compare our results with the competitive baselines. The best results are in bold.

BASELINE	20NEWS3	20NEWS4	DIGITS	CORA	CITESEER
LOGREG	76.42(0.2)	60.13(1.8)	89.56(2.0)	61.05(0.0)	63.44(0.3)
SVM	75.76(0.6)	58.20(2.0)	89.53(1.5)	61.99(0.0)	60.20(0.0)
FFNN	76.10(2.0)	58.79(2.2)	90.65(1.5)	61.12(0.3)	59.64(1.0)
GCN+RBF	76.97(1.3)	57.74(2.7)	**91.24(1.5)**	58.84(0.5)	55.94(2.9)
GCN+KNN	77.82(2.5)	61.57(1.6)	**92.64(0.4)**	66.46(0.9)	62.58(0.8)
kNN-LDS	**80.25(1.7)**	**66.18(1.3)**	**91.25(1.5)**	**69.40(1.8)**	**69.07(0.4)**
PGSSL	**79.42(1.2)**	62.84(3.7)	**92.01(1.5)**	66.76(0.7)	54.46(6.25)

Table 4.2 summarizes the accuracy results for the setting where the graph is completely missing. We compare our method with the closest competitors in terms of accuracy using the paired t-test with a significance level of $\alpha = 0.05$. We highlight in bold results that were significant.

We first note that our approach tends to compare favorably to GCN+KNN, the best baseline involving heuristically computed graphs, in 20NEWS3 and 20NEWS4. This indicates that in those cases we obtain a graph that can improve the classification performance over its heuristic counterpart. While PGSSL and the kNN-LDS algorithm proposed by Franceschi et al. [21] are on a par with 20NEWS3 and DIGITS, the latter significantly outperforms the former in three datasets, namely 20NEWS4, CORA and CITESEER. A possible reason for this difference is that kNN-LDS uses an extra set of labeled data that doubles or even triples the amount of available training data. This extra training data is used to tune the graph using a classification loss. In contrast, PGSSL uses the same training set to tune both the graph and the parameters of the classification model, potentially leading to more overfitting. Also,

PGSSL depends on the computation of edge embeddings to obtain a distribution of edges (Equation (4.4)), as opposed to *k*NN-LDS that performs a sophisticated hyperparameter search in the space of graphs directly. We observe that the classification performance of PGSSL tends to degrade on datasets with a high dimensional feature space (which implies a high dimensional embedding space) like CORA and CITESEER, suggesting a dimensionality reduction pre-processing step might be necessary to increase accuracy.

We also conduct some analysis to understand the properties of the graphs learned by PGSSL, and how they differ from those learned by *k*NN-LDS. Figure 4.2 depicts the evolution of the mean edge probability among edges depending on whether they connect points with the same or different label and whether these points are in the training, validation or test set. Notably, we see that our algorithm is able to place higher probabilities in edges connecting elements that are likely to have the same label and vice versa. This holds also in validation and test, showing that our parametric encoder is able to generalize to new edges and data points. We then look at the final distribution of edges according to whether or not they connect elements with the same label, comparing our model (Figure 4.3) and *k*NN-LDS (Figure 4.4). We see that our approach is generally better at assigning large edge probabilities to pairs of points with the same label, and again this holds both in training and in validation/test.

4.3.4 Discussion

Results in Table 4.2 show that even though our method generally improves over the supervised baselines and the SSL baselines involving heuristically computed graphs, it is still behind *k*NN-LDS. Also, we note that results in CITESEER are poor in comparison with the baselines. In this section we try to understand the reasons for this and to get insights on how to improve.

Size of training set. An important difference between our method and *k*NN-LDS is that in the PGSSL framework both the graph and the parameters of the classification model are fitted on the same training set, while *k*NN-LDS requires access to an extra set of training data in order to fit the graph separately. This is due to the nature of their algorithm: it consists of two loops, an outer loop that fits the graph and an inner loop that samples from the graph distribution and fits the parameters of a GCN model. Following this setting, the authors split the validation set in two equal parts and use one of them to supervise the graph [21]. As a way of example, in the case of the Cora dataset, the original training set has size 140, and the extra labeled set *k*NN-LDS has access to has 250 elements, resulting in a big advantage with respect to the other methods. One could train PGSSL with the same amount of labeled data for a more fair comparison.

Dimensionality of the data. PGSSL does not improve the results in CORA and CITESEER with respect to the baselines, and in the latter they are far behind the rest of the datasets. We argue that a possible reason is that its features space is high-dimensional. By design, our encoder (Equation 4.6) combines individual node embeddings into edge representations that are then fed to a neural network that computes the edge probabilities. By observing Table 4.1 and Table 4.2 we note that the more high-dimensional the feature space is, the less accurate the model results. Also, from Figure 4.2 and Figure 4.3 we note that the encoder does place more mass in edges that connect elements with the same label but with less success that in the other datasets. This suggests that a dimensionality reduction pre-processing step could improve results.

4.4 Concurrent Work in Joint Models for GSSL

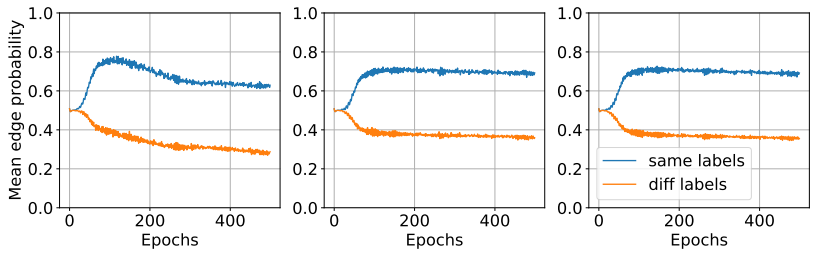
In parallel with the development of the methods described in this thesis there has been an emergence of models that try to jointly learn the graph structure along with the parameters of the graph-based SSL model, typically a GCN. Closest to our work is [21], who propose a bilevel programming formulation, consisting of an inner training error minimization objective for the classification parameters based on a sampled graph and an outer validation error minimization objective that optimizes the graph edge distribution parameters. This requires the computation of an hyper-gradient of the loss with respect to the graph, thus allowing backpropagation through its corresponding parameters. While this model is probabilistic like ours, it is not strictly end-to-end as edge parameters are treated as hyperparameters, and it is inherently transductive. Following a different route, Kazi et al. [35] introduce Differentiable Graph Module (DGM), a graph generation unit that takes as input a set of node embeddings and a set of edges (when available), and produces a new set of edges. The proposed architecture consists in a pipeline of GCN and DGM layers such that (i) the GCN layer produces node embeddings, (ii) a graph is obtained over the node embeddings using the Gaussian Kernel function with some temperature parameter to be fitted, and (iii) the node embeddings and graph are fed into the following GCN layer. The classifier is trained using the node embeddings from the final layer, while the graph parameters are updated according to a heuristics rather than in an end-to-end fashion. Finally, Chen et al. [11] propose an iterative algorithm where they simultaneously learn a graph and a data representation. The authors combine a metric learning loss, a prediction loss and a graph regularization term to fit the parameters of the graph structure and the classifier. Both of these latter models can cope with transductive and inductive regimes, but they are not probabilistic and as such

they cannot incorporate prior beliefs about the graph structure (e.g., sparsity) in a principled manner.

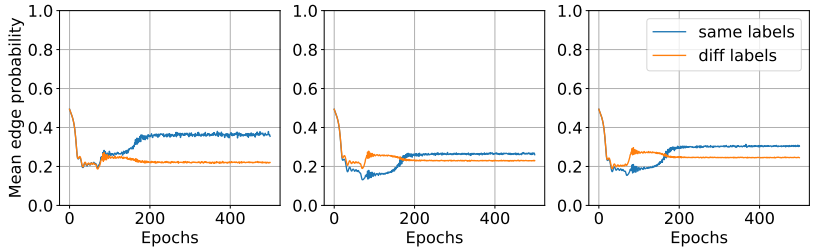
4.5 Conclusion

In this work, we presented a framework based on variational auto-encoders that simultaneously learns the parameters of a semi-supervised model and the underlying graph structure of the data. Our framework is probabilistic and can be trained in an end-to-end fashion.

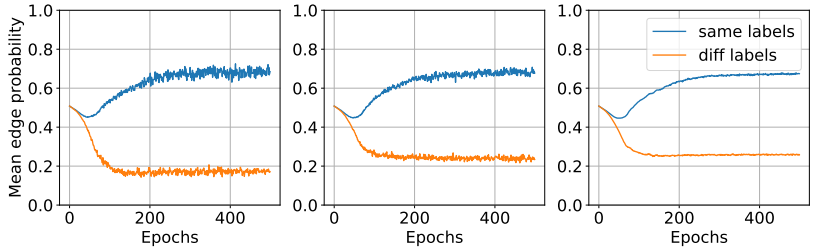
Our experiments show that our method achieve significant performance gains over the supervised baselines and semi-supervised methods based on heuristic graphs. *k*NN-LDS is a more competitive baseline based on bilevel optimization that requires an extra set of labeled data. We note that even though *k*NN-LDS achieves better performance in CORA, CITESEER and 20NEWS4, PGSSL is on a par with *k*NN-LDS in 20NEWS3 and DIGITS with a smaller training set.



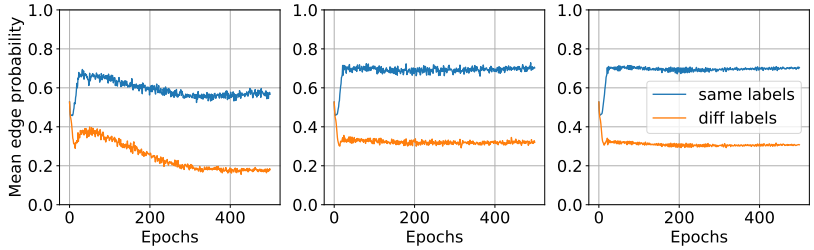
(a) CORA



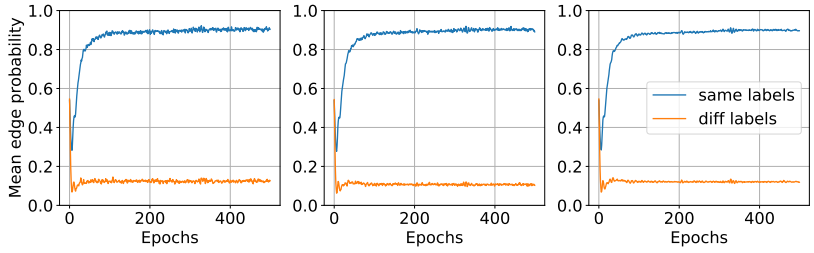
(b) CITESEER



(c) 20NEWS3

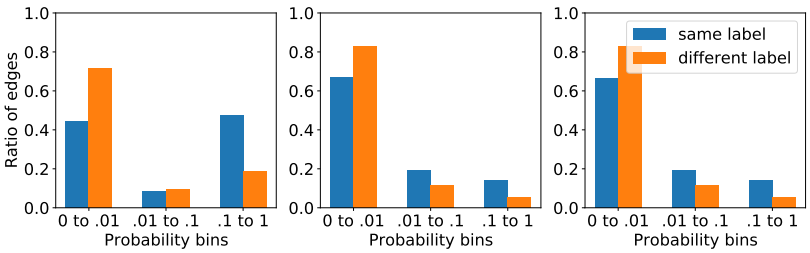


(d) 20NEWS4

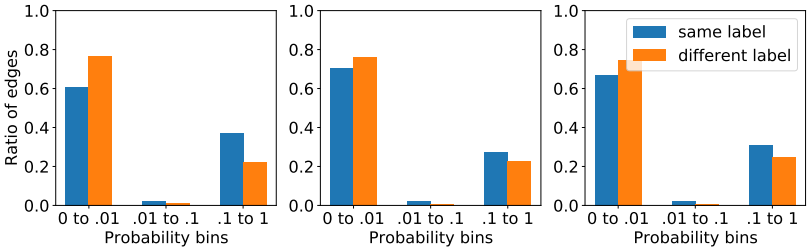


(e) DIGITS

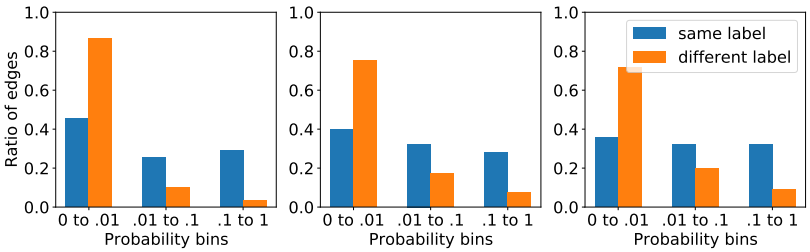
Fig. 4.2 Evolution of the mean edge probabilities for train (left), validation (center) and test (right) nodes in CORA, CITESEER, 20NEWS3, 20NEWS4 and DIGITS, shuffled with seed 0. For each node, we selected edges connecting to nodes belonging to the same class, and nodes belonging to a different class.



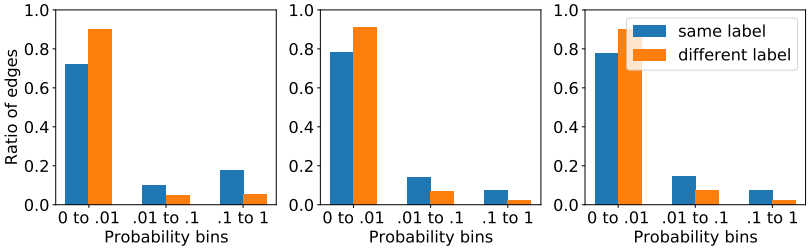
(a) CORA



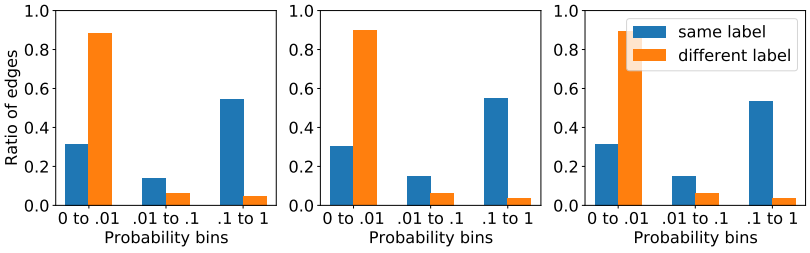
(b) CITESEER



(c) 20NEWS3

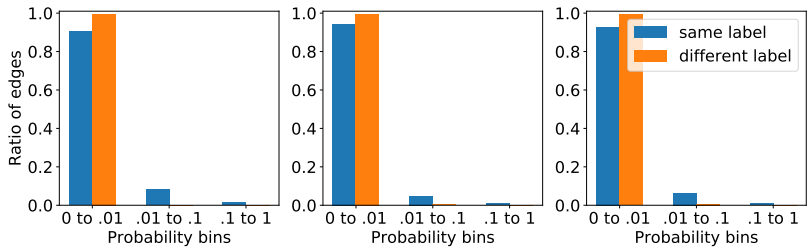


(d) 20NEWS4

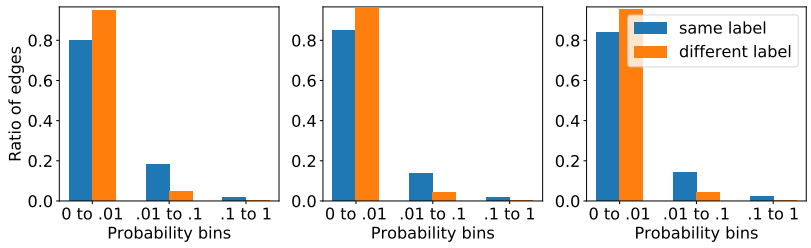


(e) DIGITS

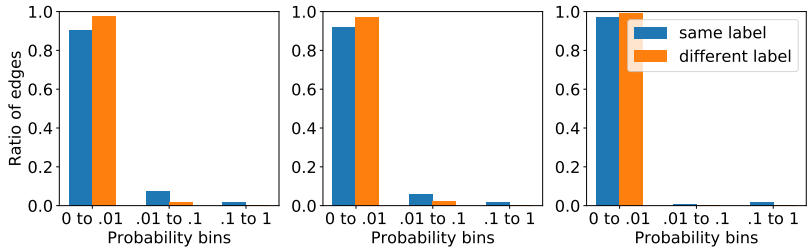
Fig. 4.3 Histograms of edge probabilities obtained with our method for train (left), validation (center) and test (right) nodes in CORA, CITESEER, 20NEWS3, 20NEWS4 and DIGITS shuffled with seed 0.



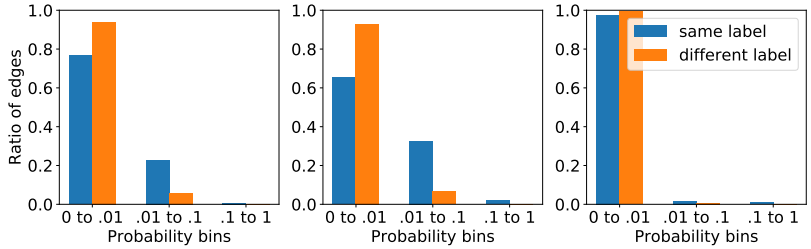
(a) CORA



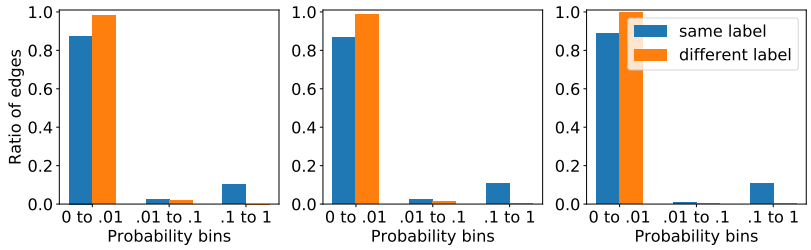
(b) CITESEER



(c) 20NEWS3



(d) 20NEWS4



(e) DIGITS

Fig. 4.4 Histograms of edge probabilities obtained with *k*NN-LDS for train (left), validation (center) and test (right) nodes in CORA, CITESEER, 20NEWS3, 20NEWS4 and DIGITS shuffled with seed 0.

Chapter 5

Conclusion

5.1 Summary

The problem of Graph-based Semi-supervised Learning in contexts where no a-priori graph is readily available is an open question in the Machine Learning community that is currently gaining a lot of attention. In this thesis we studied methods to improve the classification performance of GSSL algorithms while accounting for graph uncertainty. We addressed this question from two different perspectives.

In Chapter 3 we propose an algorithm that combines notions from Graph Learning and Metric Learning. Its main goal is to jointly infer a graph and a data representation that improves over heuristic graphs and the original data in an arbitrary GSSL algorithm. Intuitively, the data representation will cluster elements according to their class membership, and the graph will adapt to this representation. As a result, we can inject label information to the graph, thus obtaining a task-specific graph. Our experimental results suggest that the gains are especially significant for graph-based SSL algorithms that are unable to adapt the data representation (like label spreading and its variants), although we observe some gains also for GCN. We note that with our method we reduce the performance gap between more classical methods such as Label Spreading and state-of-the-art models like GCN.

In Chapter 4 we adopt a probabilistic approach based on the framework of Auto-encoding Variational Bayes proposed by [37]. The main idea is to treat the graph as a set of unobserved random variables or "latent" variables, whose distribution is to be inferred simultaneously with the parameters of a GSSL model. The learning pipeline involves taking a sample from the graph distribution, feeding it to a GSSL classification model (usually a GCN), and backpropagating the classification error through the parameters of the classification model and the graph. This model can be learned end-to-end, as opposed to what we introduced in the previous chapter. Our experiments showed mixed results. On the one hand our method

compares favorably to baselines based on heuristic graphs in two datasets, indicating the learned graph captures important information that improves the classification performance. Notably, we observe that the edge probabilities evolve to a distribution that places a high mass in edges connecting nodes belonging to the same class and vice versa. On the other hand, the more recent method proposed by [21] (kNN-LDS) outperforms our method in two datasets. A possible reason is that kNN-LDS relies on an extra set of training data whose size is non negligible (usually half of the validation set). This way the authors supervise the graph loss with a separate training set, as opposed to our method that supervises both the graph and the classification model with the same training set.

5.2 Future directions

Although we successfully tackled some of the research questions that motivated this work, some others remain open.

Other graph parameterizations. In Chapter 4 we modeled the graph as a set of independent Bernoulli random variables. Furthermore, we adopted a Stochastic Block Model [30] where an edge only depends on the nodes it connects. Other choices of model involve a less strong assumption about the edge distribution, for example, an assortative mixed membership stochastic block model [44], where two nodes not only can belong to more than one class, but edges can also depend on the class memberships of the nodes they are connecting. As we mentioned in Chapter 4, the graph parameterization we chose corresponds to a very simple Graph Neural Network where we only visit the one-hop neighborhood of each node. An alternative, more expressive graph model could involve edge embeddings that are computed based on the n-hop neighborhood of each node, with $n > 1$.

Dealing with high-dimensional data. As discussed in Section 4.3.4, our experiments in Chapter 4 suggest that the performance of the graph model degrades with high-dimensional data. A straightforward solution we did not explore is to perform a dimensionality reduction on the input data as a pre-processing step. For example, Kazi et al. [35] apply a dimensionality reduction technique called Recursive Feature Elimination to reduce the input dimension of all the datasets, keeping only 30 features. Alternatively, we could define an encoder architecture that computes a low-dimensional embedding of the data before combining elements into edges.

Closed form formulations. A path we left unexplored is that of using the tools of convex optimization to formulate and address the problem of GSSL in noisy/unavailable graph scenarios. One of the main challenges this poses is that valid graph Laplacians belongs to a small subset of Positive Semi-definite matrices, and hence they are almost impossible to find with tools like Graphical Lasso[1] [22]. Nevertheless, there exist methods in recent literature that are able to recover valid graph Laplacians from data under spectral constraints [41, 18, 40]. An interesting line of research is to extend these methods to GSSL, which could be seen as graph signal denoising in noisy/unavailable graph scenarios.

[1] Actually, finding a graph Laplacian in $\mathbb{R}^n$ is equivalent to finding a vector in the negative quadrant of $\mathbb{R}^{n(n-1)/2}$. The probability of doing that exponentially decreases as n increases.

References

[1] Adhikari, A., Ram, A., Tang, R., and Lin, J. J. (2019). Docbert: Bert for document classification. *ArXiv*.

[2] Akbik, A., Blythe, D. A. J., and Vollgraf, R. (2018). Contextual String Embeddings for Sequence Labeling. In *COLING*.

[3] Alexandrescu, A. and Kirchhoff, K. (2007). Data-Driven Graph Construction for Semi-Supervised Graph-Based Learning in NLP. In *NAACL*.

[4] Belkin, M., Niyogi, P., and Sindhwani, V. (2006). Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *JMLR*.

[5] Bellet, A., Habrard, A., and Sebban, M. (2015). *Metric Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

[6] Bronstein, M., Bruna, J., LeCun, Y., Szlam, A. D., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*.

[7] Bruna, J. (2014). Spectral networks and deep locally connected networks on graphs. In *ArXiv*.

[8] Cao, Q., Ying, Y., and Li, P. (2012). Distance metric learning revisited. In *ECML/PKDD*.

[9] Chen, J., Zhu, J., and Song, L. (2018a). Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*.

[10] Chen, M., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G. F., Jones, L., Parmar, N., Schuster, M., Chen, Z., Wu, Y., and Hughes, M. (2018b). The best of both worlds: Combining recent advances in neural machine translation. In *ACL*.

[11] Chen, Y., Wu, L., and Zaki, M. (2020). Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. In *NeurIPS*.

[12] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *CVPR*.

[13] Daitch, S. I., Kelner, J. A., and Spielman, D. A. (2009). Fitting a graph to vector data. In *ICML*.

[14] Davis, J. V., Kulis, B., Jain, P., Sra, S., and Dhillon, I. (2007). Information-theoretic metric learning. In *ICML*.

[15] Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the em algorithm plus discussions on the paper. In *JRSS*.

[16] Denis, P. and Ralaivola, L. (2017). Online Learning of Task-specific Word Representations with a JointBiconvex Passive-Aggressive Algorithm. In *EACL*.

[17] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.

[18] Dong, X., Thanou, D., Frossard, P., and Vandergheynst, P. (2016). Learning Laplacian Matrix in Smooth Graph Signal Representations. *IEEE Transactions on Signal Processing*.

[19] Everett, B. (2013). *An Introduction to Latent Variable Models*. Monographs on Statistics and Applied Probability.

[20] Fort, K. (2016). *Collaborative Annotation for Reliable Natural Language Processing: Technical and Sociological Aspects*. .

[21] Franceschi, L., Niepert, M., Pontil, M., and He, X. (2019). Learning Discrete Structures for Graph Neural Networks. In *ICML*.

[22] Friedman, J., Hastie, T., and Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*.

[23] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*.

[24] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*.

[25] Gilmer, J., Schoenholz, S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *ICML*.

[26] Globerson, A. and Roweis, S. (2005). Metric learning by collapsing classes. In *NeurIPS*.

[27] Goldberger, J., Hinton, G. E., Roweis, S., and Salakhutdinov, R. R. (2005). Neighbourhood components analysis. In *NeurIPS*.

[28] Goldstone, R. L., Medin, D., and Halberstadt, J. (1997). Similarity in context. *Memory & Cognition*.

[29] Hoi, S., Liu, W., Lyu, M., and Ma, W.-Y. (2006). Learning Distance Metrics with Contextual Constraints for Image Retrieval. In *CVPR*.

[30] Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social Networks*.

[31] Hu, J., Lu, J., and Tan, Y.-P. (2014). Discriminative Deep Metric Learning for Face Verification in the Wild. In *CVPR*.

[32] Joachims, T. (2003). Transductive learning via spectral graph partitioning. In *ICML*.

[33] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An Introduction to Variational Methods for Graphical Models. *Machine Learning*.

[34] Kalofolias, V. (2016). How to learn a graph from smooth signals. In *AISTATS*.

[35] Kazi, A., Cosmo, L., Navab, N., and Bronstein, M. (2020). Differentiable Graph Module (DGM) for Graph Convolutional Networks. In *ArXiv*.

[36] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

[37] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *ICLR*.

[38] Kipf, T. N., Fetaya, E., Wang, K., Welling, M., and Zemel, R. S. (2018). Neural Relational Inference for Interacting Systems. In *ICML*.

[39] Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[40] Kumar, S., Ying, J., de Miranda Cardoso, J. V., and Palomar, D. (2019). Structured Graph Learning Via Laplacian Spectral Constraints. In *NeurIPS*.

[41] Lake, B. M. and Tenenbaum, J. B. (2010). Discovering structure by learning sparse graph. In *CogSci*.

[42] Lawrence, S., Giles, C. L., and Bollacker, K. (1999). Digital libraries and autonomous citation indexing. *Computer*.

[43] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object Recognition with Gradient-Based Learning. In *Shape, Contour and Grouping in Computer Vision*, Berlin, Heidelberg. Springer-Verlag.

[44] Li, W., Ahn, S., and Welling, M. (2016). Scalable MCMC for Mixed Membership Stochastic Blockmodels. In *AISTATS*.

[45] Ma, J., Tang, W., Zhu, J., and Mei, Q. (2019). A flexible generative framework for graph-based semi-supervised learning. In *NeurIPS*.

[46] Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*.

[47] McNemar Quinn (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*.

[48] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. In *ICLR*.

[49] Muja, M. and Lowe, D. G. (2014). Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36.

[50] Papa, G., Bellet, A., and Clémençon, S. (2016). On Graph Reconstruction via Empirical Risk Minimization: Fast Learning Rates and Scalability. In *NeurIPS*.

[51] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *JMLR*.

[52] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *EMNLP*.

[53] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018a). Deep contextualized word representations. In *NAACL*.

[54] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer., L. (2018b). Deep contextualized word representations. In *NAACL*.

[55] Sandryhaila, A. and Moura, J. M. F. (2014). Discrete signal processing on graphs: Frequency analysis. *IEEE Transactions on Signal Processing*.

[56] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The Graph Neural Network Model. *Neural Networks*.

[57] Schultz, M. and Joachims, T. (2004). Learning a Distance Metric from Relative Comparisons. In *NeurIPS*.

[58] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008a). Collective Classification in Network Data. *AI Magazine*.

[59] Sen, P., Namata, G. M., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. (2008b). Collective Classification in Network Data. *AI Magazine*.

[60] Shuman, D., Narang, S., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*.

[61] Song, Z., Yang, X., Xu, Z., and King, I. (2021). Graph-based semi-supervised learning: A comprehensive review. *ArXiv*.

[62] Subramanya, A. and Talukdar, P. P. (2014). *Graph-Based Semi-Supervised Learning*. Morgan & Claypool Publishers.

[63] Szummer, M. and Jaakkola, T. (2002). Partially labeled classification with Markov random walks. In *NeurIPS*.

[64] Triguero, I., García, S., and Herrera, F. (2015). Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*.

[65] Tversky, A. (1977). Features of similarity. *Psychological Review*.

[66] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*.

[67] Weinberger, K. Q., Blitzer, J., and Saul, L. (2006). Distance metric learning for large margin nearest neighbor classification. In *NeurIPS*.

[68] Xing, E., Jordan, M., Russell, S. J., and Ng, A. (2003). Distance Metric Learning with Application to Clustering with Side-Information. In *NeurIPS*.

[69] Yang, F., Fan, K., Song, D., and Lin, H. (2020). Graph-based prediction of Protein-protein interactions with attributed signed graph embedding. *BMC Bioinformatics*.

[70] Ying, Y. and Li, P. (2012). Distance Metric Learning with Eigenvalue Optimization. *JMLR*.

[71] Zhang, Y., Pal, S., Coates, M., and Üstebay, D. (2019). Bayesian graph convolutional neural networks for semi-supervised classification. In *AAAI*.

[72] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2004a). Learning with local and global consistency. In *NeurIPS*.

[73] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2004b). Learning with Local and Global Consistency. In *NeurIPS*.

[74] Zhu, X. and Ghahramani, Z. (2002a). Learning from Labeled and Unlabeled Data with Label Propagation. Technical report.

[75] Zhu, X. and Ghahramani, Z. (2002b). Learning from Labeled and Unlabeled Data with Label Propagation. Technical report, Carnegie Mellon University.

[76] Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML*.

[77] Zhu, X., Lafferty, J., and Ghahramani, Z. (2018). Semi-supervised learning : from Gaussian fields to Gaussian processes. Technical report, Carnegie Mellon University.