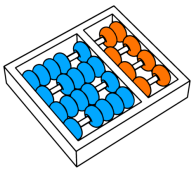
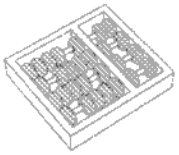


Diego Alonso Chávez Escalante

"Semi-supervised learning with graphs methods using signal processing."

"*Métodos de aprendizado semi-supervisionado com grafos usando processamento de sinais.*"

CAMPINAS
2014



University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

Diego Alonso Chávez Escalante

"Semi-supervised learning with graphs methods using signal processing."

Supervisor:
Orientador(a): **Prof. Dr. Siome Klein Goldenstein**

"Métodos de aprendizado semi-supervisionado com grafos usando processamento de sinais."

MSc Dissertation presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a Mestre degree in Computer Science.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

THIS VOLUME CORRESPONDS TO THE FINAL VERSION OF THE DISSERTATION DEFENDED BY DIEGO ALONSO CHÁVEZ ESCALANTE, UNDER THE SUPERVISION OF PROF. DR. SIOME KLEIN GOLDENSTEIN.

Este exemplar corresponde à versão final da Dissertação defendida por Diego Alonso Chávez Escalante, sob orientação de Prof. Dr. Siome Klein Goldenstein.

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

C398s

Chávez Escalante, Diego Alonso, 1988-
Semi-supervised learning with graphs methods using signal processing / Diego Alonso Chávez Escalante. – Campinas, SP : [s.n.], 2014.

Orientador: Siome Klein Goldenstein.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Aprendizado de máquina. 2. Processamento de sinais. 3. Algoritmos em grafos. 4. Redes neurais (Computação). I. Goldenstein, Siome Klein,1972-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Métodos de aprendizado semi-supervisionado com grafos usando processamento de sinais
Palavras-chave em inglês:
Machine learning
Signal processing
Graph algorithms
Neural networks (Computer science)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Siome Klein Goldenstein [Orientador]
Jacques Wainer
Renato da Rocha Lopes
Data de defesa: 27-06-2014
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Defesa de Dissertação de Mestrado em Ciência da Computação, apresentada pelo(a) Mestrando(a) **Diego Alonso Chávez Escalante**, aprovado(a) em **27 de junho de 2014**, pela Banca examinadora composta pelos Professores Doutores:

Prof(a). Dr(a). Renato da Rocha Lopes
Titular

Prof(a). Dr(a). Jacques Wainer
Titular

Prof(a). Dr(a). Siome Klein Goldenstein
Presidente

Institute of Computing /*Instituto de Computação*
University of Campinas /*Universidade Estadual de Campinas*

Semi-supervised learning with graphs methods using signal processing.

Diego Alonso Chávez Escalante[1]

June 27, 2014

Examiner Board/*Banca Examinadora*:

- Prof. Dr. Siome Klein Goldenstein (Supervisor/*Orientador*)
- Prof. Dr. Jacques Wainer
  Institute of Computing - UNICAMP
- Prof. Dr. Renato da Rocha Lopes
  Faculty of Electrical Engineering and Computing - UNICAMP
- Prof. Dr. Helio Pedrini
  Institute of Computing - UNICAMP (Substitute/*Suplente*)
- Prof. Dr. Eduardo Valle
  Faculty of Electrical Engineering and Computing (Substitute/*Suplente*)

[1]Financial support: CNPq scholarship (process 132206) 2012–2014

Abstract

In machine learning, classification problems were traditionally addressed by supervised learning algorithms, which only use labeled data for training. However, labeled data in many problem domains are really hard to collect, while unlabeled data are usually easy to collect. Also, in machine learning, only unsupervised learning is capable to learn the topology and properties of a set of unlabeled data. In order to do a classification using knowledge from labeled and unlabeled data, it is necessary to use concepts from both supervised and unsupervised learning. This type of learning is called semi-supervised learning, which has claimed to build better classifiers than the traditional supervised learning in some specific conditions, because it does not only learn from the labeled data, but also from the natural properties of unlabeled data as for example spatial distribution.

Semi-supervised learning presents a broad collection of methods and techniques for classification. Among them there is graph based semi-supervised learning, which model the problem of semi-supervised classification using graph theory. One problem that arises from this technique is the cost for training large data sets, so the development of scalable and efficient algorithms for graph based semi-supervised learning is a interesting and promising problem to deal with. From this research we developed two algorithms, one for graph construction using unsupervised neural networks; and other for graph regularization using graph signal processing theory, more specifically using FIR filters over a graph. Both solutions showed comparable performance to other literature methods in terms of accuracy.

Resumo

No aprendizado de máquina, os problemas de classificação de padrões eram tradicionalmente abordados por algoritmos de aprendizado supervisionado que utilizam apenas dados rotulados para treinar-se. Entretanto, os dados rotulados são realmente difíceis de coletar em muitos domínios de problemas, enquanto os dados não rotulados são geralmente mais fáceis de recolher. Também em aprendizado de máquina só o aprendizado não supervisionado é capaz de aprender a topologia e propriedades de um conjunto de dados não rotulados. Portanto, a fim de conseguir uma classificação utilizando o conhecimento a partir de dados rotulados e não rotulados, é necessário o uso de conceitos de aprendizado supervisionado tanto como do não supervisionado. Este tipo de aprendizagem é chamado de aprendizado semi-supervisionado, que declara ter construído melhores classificadores que o tradicional aprendizado supervisionado em algumas condições especificas, porque não só aprende dos dados rotulados, mas também das propriedades naturais dos dados não rotulados como por exemplo a distribuição espacial deles.

O aprendizado semi-supervisionado apresenta uma ampla coleção de métodos e técnicas para classificação, e um dos mais interessantes é o aprendizado semi-supervisionado baseado em grafos, o qual modela o problema da classificação semi-supervisionada utilizando a teoria dos grafos. Mas um problema que surge a partir dessa técnica é o custo para treinar conjuntos com grandes quantidades de dados, de modo que o desenvolvimento de algoritmos escaláveis e eficientes de aprendizado semi-supervisionado baseado em grafos é um problema muito interessante e prometedor para lidar com ele. Desta pesquisa foram desenvolvidos dois algoritmos, um para a construção do grafo usando redes neurais não supervisionadas e outro para a regularização do grafo usando processamento de sinais em grafos, especificamente usando filtros de resposta finita sobre o grafo. As duas soluções mostraram resultados comparáveis com os da literatura.

Acknowledgements

Finally I am finishing my master's studies. It has been more than two years of a lot of new things for me, not just new knowledge, but also new experiences. I have met new people and I have done new things, and this, this master thesis, is the product of all of that, this is the ultimate result of those two years. I have to thank so many people for this, that maybe, and just maybe I would forget to mention some of them, and not by purpose.

Firstly I have to thank my mother, Silvia, because if she had not done all the things she did for me, I would not be here, and this master thesis would have never be written. Also I have to thank the rest of my family, my father Alonso, my sister Priscila and my brother Martin, for pray for me and always wish me the best. Another special person that I have to thank is my grandmother Vicenta, she always encourage me to press forward, regardless of the situation, and maybe she is the person that has prayed more times for me. Thanks also to all my cousins, uncles and aunts, for all of their support and best wishes.

I have to thank to all of my friends, those from Arequipa and other places of Perú, who always support me in all of my decisions, regardless how illogical they were. As well to all the friends and wonderful people that I meet here, the Peruvians, Brazilians, Colombians, Bolivians, Ecuadorians, Mexicans and Spanish. The colleagues from RECOD, my friends from my house and other places, my roommates. They always cheer me up when the things went wrong, and many times they were wrong. I could not forget Carolina, she was always supporting me, praying for me, wishing me good luck, and many other things, If it was not for her maybe I would have quit everything.

Thanks also, to all my professors from Brazil and Perú, they teach me most of the things I know, special thanks to my advisor Siome Klein Goldenstein and my co-advisor Gabriel Taubin, thanks a lot for the patience, for explain me all the things that I did not understand over and over again.

I have to mention also my gratitude for the Institute of Computing, it was like my home for these two years, thank to the UNICAMP. And finally special thanks to the CNPq for the financial support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine Learning, or Pattern Recognition, is a subarea from the Intelligent Systems (IS) or Artificial Intelligence (AI) area [32]. The task of machine learning is to make the computers do things for which they were not explicitly designed to.

Traditionally machine learning can be divided in three types: supervised learning, unsupervised learning and reinforcement learning.

- Supervised Learning: the objective is to predict the type of certain group of patterns. In order to accomplish that it is already known some subset of patterns with its corresponding types. The idea is to use these already known types and the corresponding patterns for learning a rule that let us know the type of other patterns, from which we do not know their types. These types can be labels or numbers. Supervised Learning can do two task: Classification (when the types are labels) and Regression (when the types are numbers). Supervised Learning is widely use in Identification and Recognition Systems [13].
- Unsupervised Learning: the objective is to find relations among a group of patterns, and organize them by their topological distribution in the patterns space. The more common applications of unsupervised learning are clustering, outlier detection and dimensionality reduction [13].
- Reinforcement Learning: the objective is to maximize the rewards obtained from doing some specific action in some specific environment. Many applications of reinforcement learning are used in distributed agents and robotic systems [44].

Beyond these three types of machine learning there exists another one which is a combination of supervised and unsupervised learning, called Semi-Supervised Learning (SSL). It can be applied in classification, dimensionality reduction, clustering, etc [57, 7]. In Semi-supervised Learning for classification, the learning algorithms take advantage of the unlabeled data that a

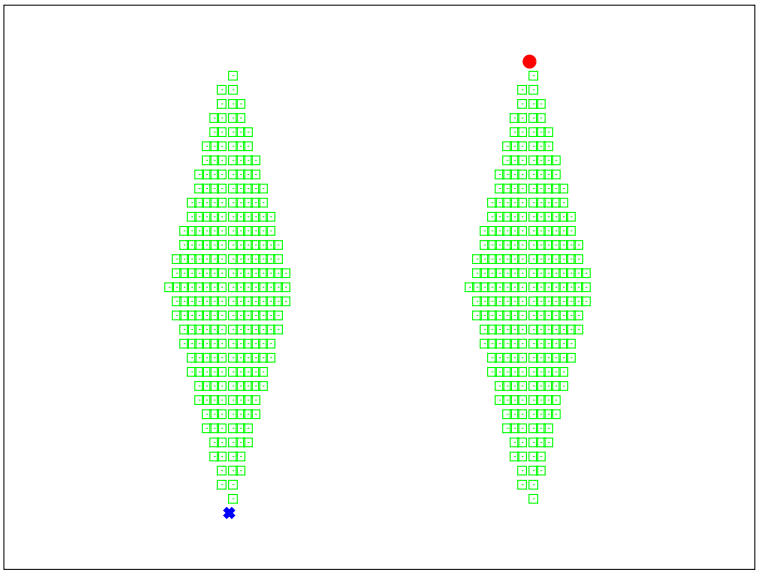
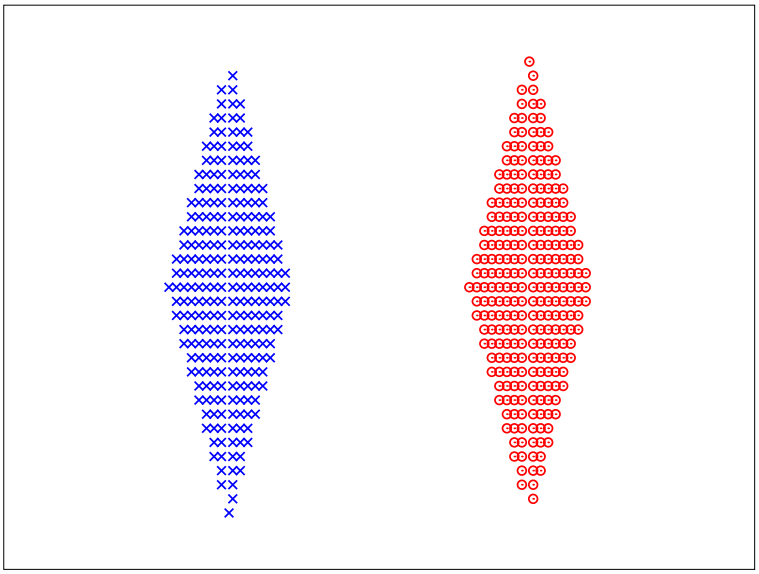


Figure 1.1: A sample data set, with two labeled data-points.

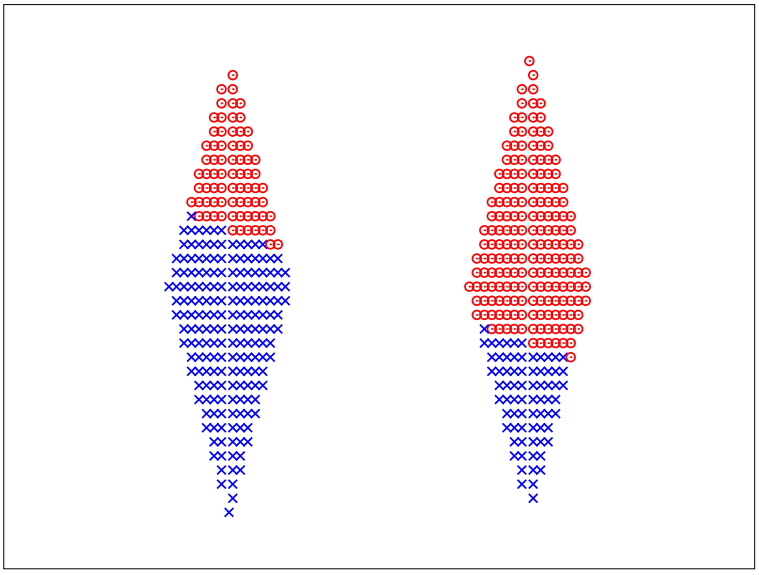
traditional supervised classifier can not use. In Figure 1.2, one can see a simple comparison of a semi-supervised learning algorithm with a supervised algorithm (KNN) over the data set of Figure 1.1. As the supervised algorithm can not use the unlabeled data-points to learn about the distribution or topology of the data set it can have the result shown in Figure 1.2(b) while the semi-supervised algorithm shows a better looking result that respects the data distribution (Figure 1.2(a)).

In most real problems, unlabeled samples are easier to get and are available in larger quantities than their labeled counterparts. Sometimes obtaining labeled data is quite difficult, it can require help from a specialist in the corresponding knowledge area, which can be very expensive. Therefore, semi-supervised learning algorithms that learn from the labeled (in very small quantities) and unlabeled data (in big quantities) can have better performance than traditional supervised algorithms in the same conditions. That conditions happen when it is not possible to obtain enough quantities of labeled data, but obtaining unlabeled data is quite easy. These conditions can be called as semi-supervised learning conditions for classification [57, 58, 55].

The semi-supervised learning area for classification has five groups of algorithms: Mixture Models, Co-Training, Self-Training, Semi-supervised Support Vector Machines and Graph based methods; the first four groups are adaptations of supervised algorithm (they will be briefly described in chapter 2) and just the fifth is semi-supervised by nature [57, 58, 55]. This master dissertation is focused on Graph based Semi-supervised learning. The graph based methods use



(a)



(b)

Figure 1.2: (a) Semi-supervised learning solution (Harmonic Regularizer). (b) Supervised learning solution (1NN algorithm).

a graph representation of the problem which give us a powerful backbone theory to use. Also the graph signal processing theory, an emerging theory that use the traditional signal processing area applied to graphs [39, 41], can be very useful in this problem.

In this Master dissertation, we propose two different methods; a graph construction technique based in an unsupervised learning neural network, and a regularization algorithm with a graph signal processing approach.

1.1 Motivation

Nowadays, machine learning applications are everywhere. We use them several times a day without knowing it. Examples of machine learning applications are spam filters, web search, face detection, etc. Also new possible applications are very promising in the software industry, turning them into rising Start-Ups. The growing numbers of machine learning applications is partially due to the rising interest in the area by new researchers, and because many existing problems can have solutions modeled with machine learning approaches.

Semi-supervised learning can help a great number of machine learning applications:

- Text categorization, in applications of spam detection. Labeled samples can be hard to obtain, because not all users like to keep labeling the mails they got as spam or not spam. But unlabeled samples are as common as the quantity of mails received by the user. So the semi-supervised learning condition is fulfilled.
- Natural language parsing, in order to recognize text. It is imperative to train a good parser algorithm, for this is needed pairs of sentences know as treebanks, these treebanks are built by linguistic specialists. Since it takes considerable time to build them, treebanks are difficult and expensive to obtain in large quantities. But the unlabeled samples for these problems (sentences) are just everywhere. Again this problem satisfy the semi-supervised learning condition.
- Video Surveillance, in this problem the samples are the video frames, and the labels are the kind of objects that appear in each frame. Labeling all the video frames is a slow and tedious task, so the number of labeled samples is very low, but the unlabeled samples are a lot, because video cameras can record all day long.
- 3D protein's structure prediction, the DNA sequences are the data-points in this specific problem, and the labels are the 3D protein's structure. For a specialist in the domain, this task could take months, so labeled samples are very difficult to find, but there are a lot of DNA data sets available for research.

There are many more applications beyond the ones mentioned above. As it was said before, any problem that satisfies the semi-supervised learning condition is ready to be solve with a semi-supervised approach. Also, Graph based methods are good in classification tasks where

patterns that are very similar between them belong to the same class, and those that are very different tend to be of different classes. This supposition is reasonable for many real problems.

1.2 Problem definition

The problem to be addressed in this master dissertation is:

Given a data set X of $l + u$ patterns ($|X| = l + u$), where $\forall\ x \in X$, $x \in \mathbb{R}^n$ ($X \subset \mathbb{R}^n$), with l labeled points and u unlabeled points ($l \ll u$). It is also given a set Y_l of labels of size l ($|Y_l| = l$), $Y_l \subset Y$, where Y is the set of all the labels of X (including the known labels Y_l and the unknown labels Y_u), $|Y| = l + u$, $\forall\ y \in Y$, $y \in \{0, 1\}$. We aim to predict the labels' set Y_u ($|Y_u| = u$), $Y_u \subset Y$ and $Y_l^{\complement} = Y_u$ ($Y_l \cup Y_u = Y$) (for each $x \in X$ exist only one $y \in Y$). As premise for this prediction, $\forall\ x_i,\ x_j,\ x_h \in X$ if x_i and x_j have the same labels (that is $y_i = y_j$), also if x_i and x_h have different labels ($y_i \neq y_h$), then the similiarity of x_i and x_j is higher than the similarity of x_i and x_h.

The similarity measure is a function $s(d, x_a, x_b) \in [0, 1]$, if it is 1 (or the maximum possible value depending of the similarity measure) then $x_a = x_b$ and 0 if the opposite happens; d is a distance function, the similarity has to be based in a distance function so if $d(x_a, x_b) = 0$ then $s(d, x_a, x_b) = 1$. Thus if $y_i = y_j$ and $y_i \neq y_h$ then $s(d, x_i, x_j) > s(d, x_i, x_h)$ (also $d(x_i, x_j) < d(x_i, x_h)$).

In order to accomplish this prediction of labels, it is going to be used the semi-supervised learning with graphs theory, for developing a method that is effective and scalable to solve this problem. Using the aid of graph signal processing theory and unsupervised neural networks.

Chapter 2

Semi-Supervised Learning

Semi-Supervised Learning (SSL) is a hybrid between supervised learning and unsupervised learning, where labeled data is exploited together with unlabeled data in order to do classification, regression, clustering and dimensionality reduction. This dissertation focus on semi-supervised learning for classification. In semi-supervised classification, the input data set X can be of three different types:

- **The Labeled Data** set $X_l = \{x_1, x_2, ..., x_l\}$ of size l ($|X_l| = l$); whose labels Y_l are already known before the semi-supervised learning takes place ($(X_l, Y_l) = \{(x_1, y_1), (x_2, y_2), ..., (x_l, y_l)\}$).

- **The Known Unlabeled Data** set $X_u = \{x_{l+1}, x_{l+2}, ..., x_{l+u}\}$ of size u ($|X_u| = u$); whose labels Y_u are not known before the semi-supervised learning takes place ($(X_u, Y_u) = \{(x_{l+1}, y_{l+1}), (x_{l+2}, y_{l+2}), ..., (x_{l+u}, y_{l+u})\}$). It is supposed that these labels have to be discovered by the semi-supervised learning algorithm. The X_u set is a numerous set, and this entire set is known before the algorithm takes place.

- **The Unknown Unlabeled Data** set $X_k = \{x_{l+u+1}, x_{l+u+2}, ..., x_{l+u+k}\}$ of size k ($|X_k| = k$); whose labels Y_k are not known before the semi-supervised learning takes place, like the data-points themselves ($(X_k, Y_k) = \{(x_{l+u+1}, y_{l+u+1}), (x_{l+u+2}, y_{l+u+2}), ..., (x_{l+u+k}, y_{l+u+k})\}$). This set becomes known only after the learning.

Semi-Supervised learning for classification can be of two types depending of its input data, transductive or inductive. The description and differences between these two types are explained below.

2.1 Transductive Learning

Transductive learning works with the (X_l, Y_l) and X_u sets as inputs. It trains a function $f : \mathbb{R}^n \longrightarrow L$, where L is the label set, for binary classification $L = \{0, 1\}$. The parameters of f are learned from $X_l \cup X_u$ and Y_l in order to predict the labels Y_u of the X_u set, respecting the already known Y_l labels [57]. The process is shown in Figure 2.1. The transductive learning can be divided in two steps, one for placing the data or build some structure from it (initialization), and other for the labeling process (transductive classification).

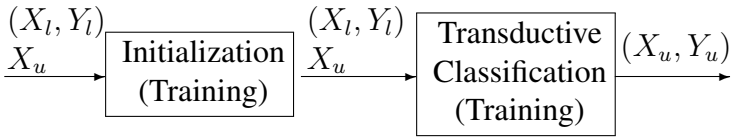


Figure 2.1: Transductive Learning Process.

2.2 Inductive Learning

Inductive learning works with (X_l, Y_l), X_u and X_k sets as inputs at different phases of the process. It trains a function $f : \mathbb{R}^n \longrightarrow L$. It learns f from $X_l \cup X_u$ and Y_l, in the training process it discover the labels Y_u (Transductive Learning), and then it can do the inductive step, making predictions on future data X_k, discovering the labels of Y_k. It is assumed that the elements of set X_k can be analyzed one by one, and k is not known [57]. Figure 2.2 shows the inductive learning process. The semi-supervised inductive learning needs a transductive learning before it can work. The inductive learning can vary its learning parameters while it its classifying new data-points x ($x \in X_k$), this is why it is called learning.

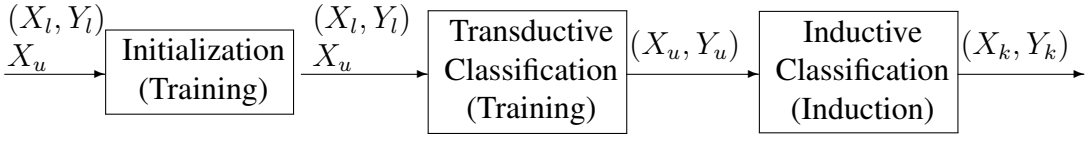


Figure 2.2: Inductive Learning Process.

2.3 Semi-supervised methods

The semi-supervised learning for classification literature categorize all of its methods in five groups, regardless if they are inductive or transductive, see Figure 2.3. Semi-supervised algorithms make assumptions about the distribution and topology of the data related to each label. That is one can assume that the data-points related to one specific class have some properties in common, the assumption are the properties that are assumed to be true for each problem. Generally semi-supervised learning methods, in order to be applied to a specific problem, the involved data set should meet the following three assumptions [7]:

- **The semi-supervised smoothness assumption**. If two points x_1, x_2 in a high-density region are close, then the corresponding labels y_1, y_2 should be equal.
- **The cluster assumption**. If x_1 and x_2 are in the same cluster, they should have equal labels.
- **The manifold assumption**. The high dimensional data lie on a low dimensional manifold.

Each semi-supervised method type assumes some specific assumptions related to those mentioned above.

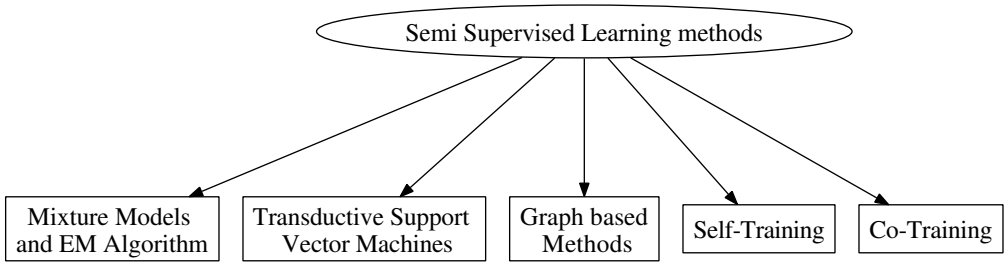


Figure 2.3: Semi-supervised methods.

2.3.1 Self-training

This is a semi-supervised adaptation from a supervised algorithm. The idea is to train a traditional supervised classifier f_{st} with the available labeled data X_l. After that it is picked some subset X_s from X_u ($X_s \subset X_u$). Then X_s is classified by f_{st}, after that f_{st} is trained again with the new labeled data $X_l \cup X_s$. Later a new unlabeled subset is picked from $X_u - X_s$, and it is

classify by the new f_{st}. The process is repeated until there is no more unlabeled data [57, 37, 53].

The assumption of self-training is that its own predictions are relatively correct, at least the high confidence ones; for example, this would be the cases when the data forms highly separable clusters (cluster assumption).

2.3.2 Co-training

It is a wrapper method (as Self-training) that adapts a supervised classification algorithm into a semi-supervised one. First each data point's descriptor has to be split in two parts which we are going to call as representations so each $x \in X$ would be expressed as $x = [x^{(1)}, x^{(2)}]$, being $x^{(1)}$ its first representation and $x^{(2)}$ its second representation. Also there are two classifiers $f^{(1)}$ and $f^{(2)}$, $f^{(1)}$ would work with the first representation of the data-points and $f^{(2)}$ would work with the second representation of the data-points. Each classifier is trained with X_l, after that they classify the data in X_u (using its corresponding data representation). then each classifier picks its k most confident predictions ($X_{f^{(1)}}$ and $X_{f^{(2)}}$). Then $X_{f^{(1)}}$ is trained again with $X_l \cup X_{f^{(2)}}$ and $X_{f^{(2)}}$ with $X_l \cup X_{f^{(1)}}$. The process is repeated until there is no more unlabeled data [57, 26, 22, 6, 28].

Co-training makes the following assumptions: each data point can be divided in two coherent different representations; each representation alone is sufficient to make a good classification; and the two representations are conditionally independent given the class label, that is if we know the true label from one representation, knowing the other representation does not change anything.

2.3.3 Mixture Models and EM

This is the Semi-supervised adaptation of the traditional Mixture Models combined with an expectation maximization algorithm. It is perhaps the oldest semi-supervised learning method in the literature. It assumes a generative model $p(x, y) = p(y)p(x|y)$, where x is a data-point , y the corresponding label and $p(x|y)$ is an identifiable mixture distribution. With big quantities of unlabeled data, the mixture components can be identified. Then with just one or very few labeled points per class, we can fully determined the mixtures distributions [57, 10, 36].

The assumption behind Semi-supervised Mixture Models is that the data-points actually come from the mixture model distribution, where the prior probability $p(y)$, the conditional probability $p(x|y)$ and the number of components are the real ones (manifold assumption).

2.3.4 Semi-supervised Support Vector Machines

These are the semi-supervised variation of the traditional supervised support vector machines. They were originally call Transductive Support Vector Machines (TSVMs), but now are commonly called Semi-supervised Support Vector Machines (S3VMs). S3VM uses labeled and unlabeled data, to maximize the geometric margin not just by using the labeled data X_l but also the unlabeled data X_u. S3VM finds the hyperplane that separates better the classes of X_l and at the same time that best divides X_u, which is not the same one of the supervised version [51, 3, 8]. See Figure 2.4 and 2.5 for a comparative example.

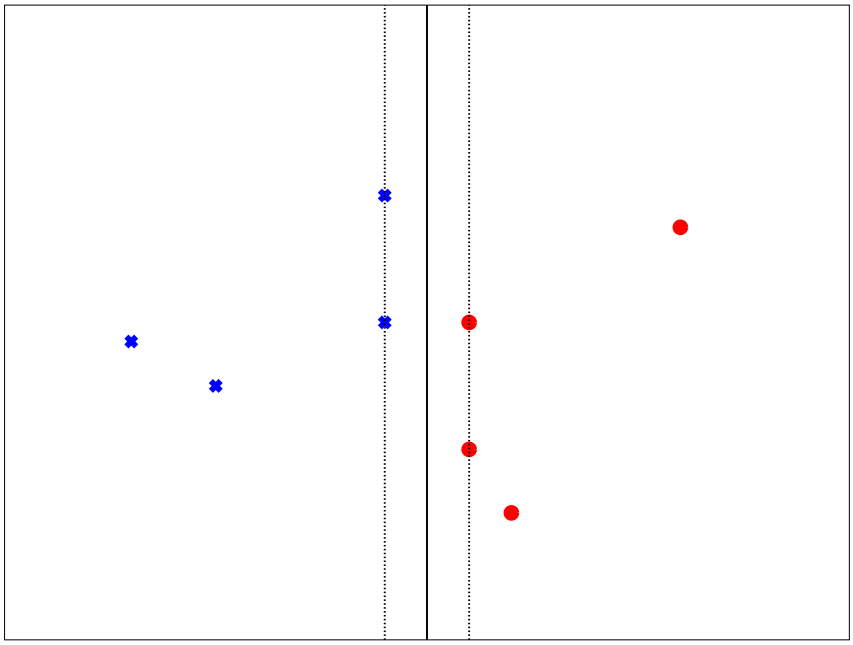


Figure 2.4: Traditional Support Vector Machine (SVM) decision boundary.

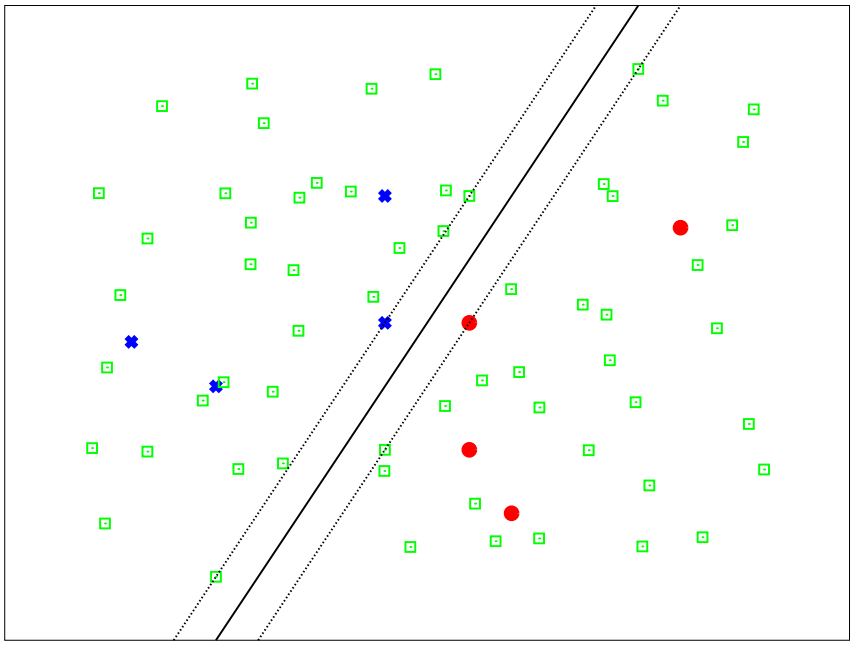


Figure 2.5: Semi-supervised Support Vector Machine (S3VM) decision boundary.

The Semi-supervised Support Vector Machines" assumption is that the classes are well separated, and that the decision boundary falls into a low density region in feature space (cluster assumption).

Chapter 3

Semi-Supervised Learning with Graphs

Semi-Supervised Learning algorithms with graphs are transductive by nature, but there are also algorithms for inductive classification [57, 58]. The idea is to build a graph $G = (V, E)$ from the input data-points ($X_l \cup X_u$, depending of the type of graph the number of vertices would be $|V| = l + u$), and then propagate the information from the labeled data X_l to the unlabeled data X_u over the graph (transductive learning). Figure 3.1 shows an already built graph, propagating the information of its vertices.

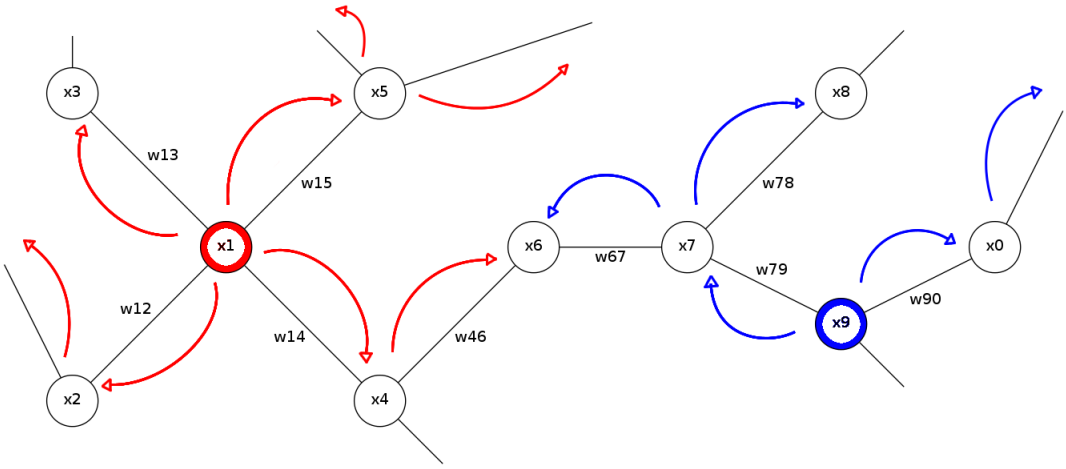


Figure 3.1: Label propagation along the graph, from the labeled data (x_1 and x_9) to the unlabeled data.

Most semi-supervised graph based methods have three steps:

- **Graph Construction**, done from the input data-points.
- **Graph Regularization** or Transductive Classification, when the labels are propagated along the graph, from the labeled to the unlabeled data.
- **Graph Induction** or Inductive Classification, when new data-points are classified one by one.

The schemes of Transductive learning and Inductive learning in Semi-supervised learning with graphs are shown in Figures 3.2 and 3.3 respectively. In this work, we focus just in transductive learning with graphs.

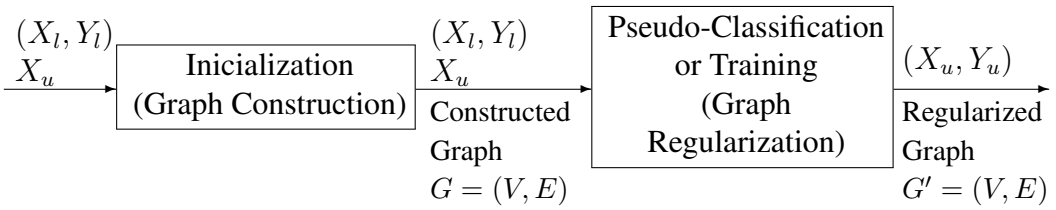


Figure 3.2: Transductive Learning with Graphs.

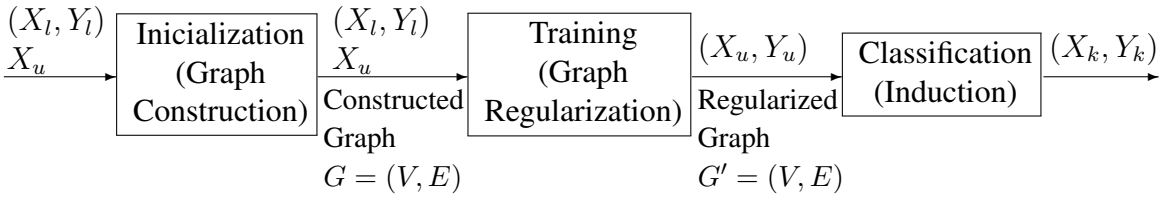


Figure 3.3: Inductive Learning with Graphs.

3.1 Graph based method's assumption

The Graph based methods assume that if two data-points are close to each other, they should be of the same class, and if they are far away, they should be in different classes (semi-supervised smoothness assumption). In other words, it means that labels' variation should be smooth with respect to the built graph. That is, in a similarity graph, if two vertices are connected with a strong edge, large weight, these two vertices tend to have the same label. Also if there exists an edge with a very low weight, in contrast with many of the other edges, the vertices adjacent to it should have different labels [57].

Spectral graph theory give us a precise notion of smoothness over a graph. In order to show that, it is necessary to first introduce the graph Laplacian. Given a graph $G = (V, E)$ represented by its weighted matrix W of size $l + u \times l + u$, the unnormalized graph Laplacian Δ is given by Equation 3.1, where the diagonal matrix D of size $l + u \times l + u$ is shown in Equation 3.2.

$$\Delta = D - W \tag{3.1}$$

$$D_{ii} = \sum_{j=1}^{l+u} w_{ij}, i = 1, 2, ..., l + u \tag{3.2}$$

Also there is a normalized version of the graph Laplacian, which is shown in Equation 3.3, where I is the identity matrix.

$$\Delta' = D^{-1/2}\Delta D^{-1/2} = I - D^{-1/2}WD^{-1/2} \tag{3.3}$$

We have just introduced the graph Laplacian and its normalized version. Now lets talk about eigenvectors and eigenvalues. A vector ϕ is an eigenvector of a square matrix M if Equation 3.4 holds, where λ is the associated eigenvalue. Also if ϕ is an eigenvector, $c\phi$ is another eigenvector of M for any constant $c \neq 0$. But we will focus on unit length eigenvectors ($\|\phi\| = 1$).

$$M\phi = \lambda\phi \tag{3.4}$$

Spectral graph theory is concerned with the eigenvalues and eigenvectors of the graph. These pairs of eigenvectors and their corresponding eigenvalues are called graph spectrum.

In spectral graph theory the graph is commonly represented by its unnormalized Laplacian Δ or its normalized version Δ'. The unnormalized Laplacian presents the following properties:

- A graph of with $l + u$ vertices would have a graph spectrum of $l + u$ eigenvalues and eigenvectors pairs $(\lambda_i, \phi_i)_{i=1}^{l+u}$. Some eigenvalues can have the same value, and all the eigenvectors are orthogonal between them, that is $\phi_i^\top \phi_j = 0$ for any pair $i \neq j$.
- The Laplacian matrix Δ can be decomposed in a weighted sum of outer products of its eigenvectors, as Equation 3.5 shows.

$$\Delta = \sum_{i=1}^{l+u} \lambda_i \phi_i \phi_i^\top \tag{3.5}$$

- The Laplacian 's eigenvalues are non-negative real numbers, and can be sorted as $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{l+u}$.

From these we can say that if a graph has k connected components it also has k eigenvalues of zero value ($\lambda_1 = ... = \lambda_k = 0$). Also the corresponding eigenvectors are constant on individual connected components and zero else where.

3.2 Graph construction

The first step in the semi-supervised learning process is the graph construction. The graph is built from the input data-points; it has to represent the similarities among the data-points, describing its distribution and topology in some way. The graphs can be completely connected or sparse. The literature points out that in practice completely or fully connected graphs have worse results than sparse graphs, also the sparse graphs are cheaper in space and faster in training time. The edges weights of the graph have to represent the similarity between the vertices. In sparse graphs the edges can have no weight, because one can assume that each vertex connects only with the vertices that have some high similarity with it.

In most graph construction algorithms, a graph $G = (V, E)$ is build from X_l and X_u, where each $x \in X_l \cup X_u$ becomes a vertex $v \in V$, and the weight w_{ij} of each edge $e_{ij} \in E$ adjacent to the vertices v_i and v_j represents the similarity between x_i and x_j (the corresponding labels for x_i and x_j would be y_i and y_j).

The most used graphs in semi-supervised learning with graphs are:

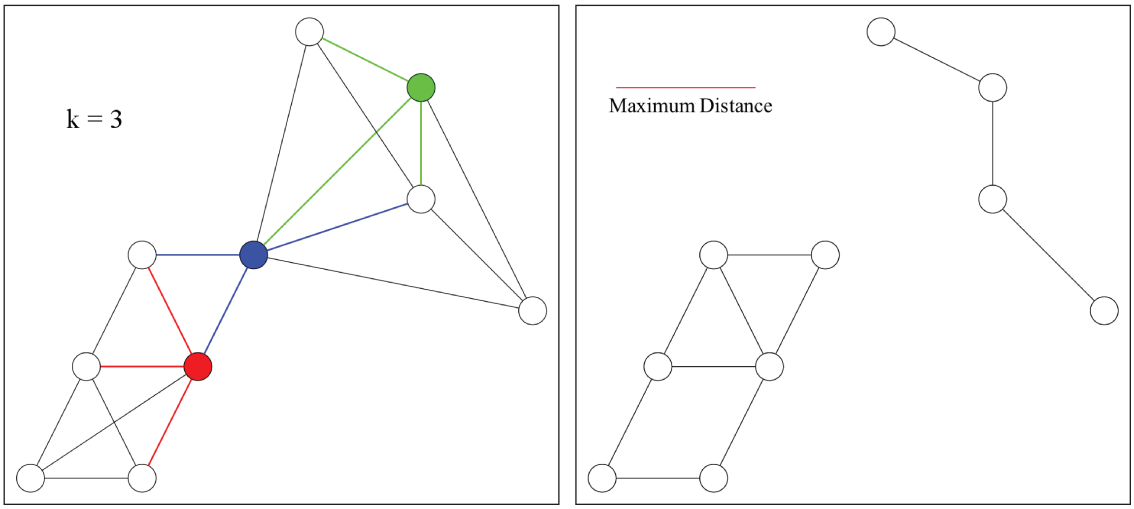


Figure 3.4: Two different sparse graphs from the same data-points using euclidean distance for its construction, (a) k-graph with $k = 3$, (b) ϵ-graph with $e = Maximum\ Distance$.

3.2.1 Full-Graphs

This kind of graph is a completely connected graph, where each vertex represents one individual data-point, and each vertex is connected to all the other vertices, the weight of each edge is a real number, normally between 0 and 1, if the weight is closer to 1 the vertices are more similar than if the weight is closer to 0.

3.2.2 k-Graphs

This is a kind of sparse graph were each vertex also represents one individual data-point, but each vertex is connected to its k nearest neighbors. In this kind of graph it is worth to point out that the degree of each vertex it is not always k, it can vary depending of the distribution of the input data-points.

3.2.3 ϵ-Graphs

In this kind of graph, each vertex connects to all the vertices that are at a distance equal or less than a real value ϵ. As the ϵ value increases, the graph tends to be a complete Full-graph.

Figure 3.4 shows two different graphs (a k-graph and a ϵ-graph) built from the same data-points.

There are also more specialized construction algorithms in the literature that generate other type of graphs [52, 1, 12, 11].

3.2.4 Similarity Measures in Graphs

In this context a similarity measure tells if two vertices are similar to each other. If it is close to 1 then the vertices should be very similar (if it is 1 the vertices are exactly equal), and if it is close 0 then the vertices should be very different (see section 1.2). The similarity measure in a weighted graph can be of different types. The two most common ones in the literature are:

- exp-weighted graphs; these graphs use as weights the values expressed by the exponential function of Equation 3.6, where the weight w_{ij} between the vertices i and j is a continuous value between 0 and 1, $d(i,j)$ is a distance function (popularly the Euclidean distance) between data-points i and j, and σ is the hyper-parameter that controls the decay rate of the exponential function.

$$w_{ij} = e^{\frac{-d(i,j)^2}{\sigma^2}} \tag{3.6}$$

- $tanh$-weighted graphs; these graphs use as weights the values expressed by the hyperbolic tangent function of Equation 3.7, where σ_1 and σ_2 are hyper parameters that control the function's slope and cut off, respectively. If $d(i,j) \gg \sigma_2$ then $w_{ij} \approx 0$, and if $d(i,j) \ll \sigma_2$ then $w_{ij} \approx 1$.

$$w_{ij} = \frac{tanh(\sigma_1(d(i,j) - \sigma_2)) + 1}{2} \tag{3.7}$$

3.3 Graph regularization

Transductive learning in Graph based Semi-supervised learning is also called as graph regularization. The idea is to propagate the labels from the labeled vertices to the unlabeled vertices, respecting the Semi-supervised learning with graphs assumption. We look for a function f ($f : V \rightarrow \mathbb{R}$) over the graph that correctly assign labels to all the vertices, respecting two conditions:

- Labeled data have to remain with the same label at the end of the regularization step, in other words minimize a loss function that sternly penalize the variation of labels of labeled data, after and before the regularization step.

- Labels' transitions over the graph have to be smooth (as the graph based methods assumption says), minimizing an energy function (the regularizer) over the graph (example: $R(f) = \sum_{e_{ij} \in E} w_{ij}(f(x_i) - f(x_j))^2$) ; that should keep vertices connected by high-similarity edges with the same label [57, 58, 55]. If two adjacent vertices have different labels, the distance between them has to be the largest possible one. In a similarity graph, this edge should have the smallest possible weight, and by this if an edge has a high weight, the adjacent vertices should have the same label.

One interpretation of the graph regularization step is as the solution of the minimization problem of Equation 3.8. Most of Graph based Semi-supervised Learning are variations of this equation.

$$\arg\min_{f} \left(\overbrace{Q(f)}^{loss\ function} + \underbrace{R(f)}_{regularizer} \right) \tag{3.8}$$

Another way to see the regularization process is as a special case of the Mincut problem [4, 5], where, given a similarity graph $G = (V, E)$, we want to find the edges that have the smallest values, in order to delete them and divide the graph in two different connected components. And so the vertices within the same connected component should have the same label, and the vertices in different connected components should have different labels.

The literature presents many ways to carry out the regularization. Like the harmonic regularizer [57, 56], other methods are of iterative nature. They use label propagation algorithms that would converge to a solution in a determined number of iterations, but this could take a while or never happen [58, 27]. In this work, we do not examine iterative label propagation algorithms because of its time consuming nature. Following, we present two classical regularization methods from the literature.

3.3.1 Harmonic regularizer

A Harmonic function has the same values as given labels on the labeled data, and the value assigned to each unlabeled vertex is the weighted average of its adjacent vertices' values (weighted average property), see Equation 3.9.

$$f(x_i) = y_i, \quad i = 1, 2 \ldots l.$$

$$f(x_i) = \frac{\sum_{j=1}^{l+u} w_{ij} f(x_i)}{\sum_{j=1}^{l+u} w_{ij}}, \quad i = l+1, l+2 \ldots l+u. \tag{3.9}$$

The Harmonic regularization scheme and the loss function minimization are shown in Equation 3.10, being the first part of the Equation the loss function and the second one the regularizer. As f can accept real values in between -1 and 1, being -1 one class and 1 the other (thresholding at zero) [57, 56].

$$\underset{f:f(x)\in\mathbb{R}}{\arg\min} \infty \sum_{i=1}^{l} (y_i - f(x_i))^2 + \sum_{i,j=1}^{l+u} w_{ij} (f(x_i) - f(x_j))^2 \tag{3.10}$$

With the graph Laplacian Equation 3.10 can be redefined as is shown in Equation 3.11.

$$\underset{f:f(x)\in\mathbb{R}}{\arg\min} \infty \sum_{i=1}^{l} (y_i - f(x_i))^2 + f^\top \Delta f \tag{3.11}$$

If the graph Laplacian matrix is redefined as in Equation 3.12, with labeled ($_l$) and unlabeled ($_u$) data-points well organized, and solving the optimization Equation on 3.11, it can be obtained the Equation 3.13, where f is directly defined in closed form.

$$\Delta = \begin{bmatrix} \Delta_{ll} & \Delta_{lu} \\ \Delta_{ul} & \Delta_{uu} \end{bmatrix} \tag{3.12}$$

$$\begin{aligned} f_l &= Y_l \\ f_u &= -\Delta_{uu}^{-1} \Delta_{ul} Y_l \end{aligned} \tag{3.13}$$

3.3.2 Global and Local consistency

The methods based in global and local consistency use a normalized version of the Laplacian in their regularizer. Equation 3.14 shows the normalized regularizer form of the regularizer of Equation 3.10 [54].

$$\frac{1}{2}\sum_{i,j=1}^{l+u} w_{ij}\left(\frac{f(x_i)}{\sqrt{D_{ii}}} - \frac{f(x_j))}{\sqrt{D_{jj}}}\right)^2 = f^\top D^{-1/2}\Delta D^{-1/2} f = f^\top \Delta' f \tag{3.14}$$

Then the minimization scheme with the loss function would be as in Equation 3.15.

$$\underset{f:f(x)\in\mathbb{R}}{\arg\min} \infty \sum_{i=1}^{l}(y_i - f(x_i))^2 + f^\top \Delta' f \tag{3.15}$$

3.4 Graph Methods for Fast Computation

As most Graph based Semi-supervised algorithms need to process the whole graph (such as calculating the graph Laplacian), they tend to be very expensive in terms of memory space, processing time and complexity. Also the graph construction step (depending of the type of graph) can be a very slow process. These problems are reflected even in not too large data sets (of no more than 2000 data-points).

Graph representation in the data structure is very important when thinking about memory space. It is very common to use matrices (sparse matrices when the graph is not a completely connected one), because it facilitates the graph Laplacian calculations, as it is represented with another matrix. Other expensive steps, in terms of memory space and processing time, are the matrix inversions, matrix multiplications and matrix spectral decomposition. For example, if a semi-supervised classification problem has a data set of ten thousand data-points, there would be matrices of 10000×10000 (one hundred million elements!), even if we are working with sparse matrices the number of elements of this matrix is just excessive.

In order to avoid these expensive steps (see Figure 3.5) in the semi-supervised learning with graphs process, the research community has put special attention in graph methods for fast computation [55, 17, 27, 14, 45]. We are going to present the most relevant graph methods for fast computation in the literature, describing the mathematical nature and the algorithmic procedure.

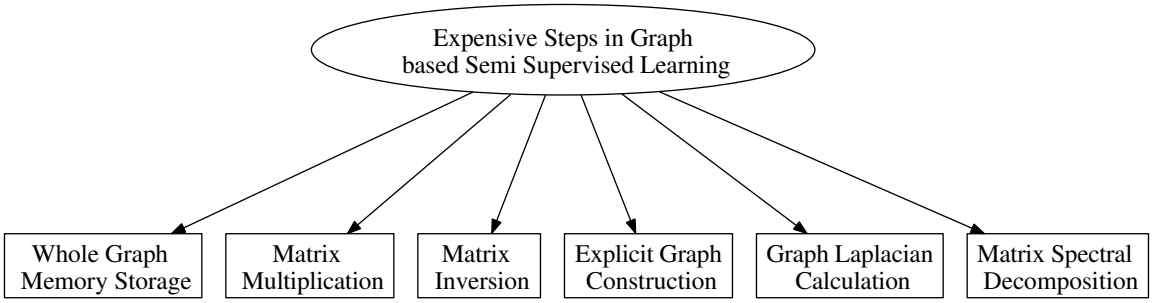


Figure 3.5: Expensive Steps in Graph based SSL.

3.4.1 Krylov Subspace Iteration and MINRES Algorithm

This is a fast computation method that minimizes the error function $E(Y_u)$ (regularization scheme) of Equation 3.16 [27].

$$E(Y_u) = \frac{1}{2}\left(\sum_{i,j\in X_l} w_{ij}(y_i - y_j)^2 + 2\sum_{i\in X_u, j\in X_l} w_{ij}(y_i - y_j)^2 + \sum_{i,j\in X_u} w_{ij}(y_i - y_j)^2\right) \quad (3.16)$$

Using matrix notation, Equation 3.16 can be rewritten as in Equation 3.17. Differentiating and equating to zero Equation 3.17 leads to the closed form solution shown in Equation 3.18 as a system of linear equations.

$$E(Y_u) = Y_u^\top \Delta_{uu} Y_u - 2Y_u^\top W_{ul} Y_l + Y_l^\top \Delta_{ll} Y_l \quad (3.17)$$

$$\Delta_{uu} Y_u = W_{ul} Y_l \quad (3.18)$$

Directly solving the system in Equation 3.18 would lead to a $O(u^3)$ complexity algorithm. One way to reduced this complexity is by using the fixed point updates of Equation 3.19 (reducing the complexity to $O(u^2)$ in the average case), but it is possible that this update scheme would take too many iterations to converge.

$$Y_u^{t+1} = D_{uu}^{-1}[W_{uu} Y_u^{(t)} + W_{ul} Y_l] \quad (3.19)$$

Equation 3.17 can be solved by approximating with an iterative method, like the MINRES algorithm [35], which uses Krylov subspace iterations to find the solution. In Krylov subspace methods is used the history of what has been already learned in every iteration to make an approximation to the solution of the system of linear equations, projecting some M-dimensional space into a lower dimensional space [38, 42, 27].

Given the graph Laplacian matrix and the vector $b = W_{ul}Y_l$, the corresponding Krylov matrix is denoted in Equation 3.20.

$$Kr^{(t)} = [b \;\; \Delta_{uu}b \;\; \Delta_{uu}^2 b \; ... \; \Delta_{uu}^t b] \tag{3.20}$$

To find an estimate of $Y_u^{(t)}$ at some iteration t, it can be done a projection onto the Krylov subspace (the spaces spanned by the column vectors of matrix $Kr^{(t)}$). As t increases in $\Delta_{uu}^t b$, it is converging to the eigenvector corresponding to the largest eigenvalue of Δ_{uu}. Kr is a poorly conditioned matrix, thus the algorithm uses a well-conditioned orthogonal matrix $Q^{(t)} = [q^{(1)} \cdots q^{(t)}]$ that spans the same space as the matrix $Kr^{(t)}$ (of t columns). In order to find this matrix Q, we can use the Gram-Schmidt process, with the recurrence of Equation 3.21, where $\beta^{(t)}$ is a normalizing constant at iteration t and $\alpha^{(t)}$ is given by Equation 3.22.

$$\beta^{(t)}q^{(t+1)} = \Delta_{uu}q^{(t)} - \beta^{(t-1)}q^{(t-1)} - \alpha^{(t)}q^{(t)} \tag{3.21}$$

$$\alpha^{(t)} = {q^{(t)}}^{\top}\Delta_{uu}q^{(t)} \tag{3.22}$$

Equation 3.22 is the Lanczos iteration [25], which produces a Schur decomposition (Equation 3.23) [34], where $\tilde{H}^{(t)}$ is a tridiagonal Hessenberg matrix (Equation 3.24).

$$\Delta_{uu}Q^{(t)} = Q^{(t+1)}\tilde{H}^{(t)} \tag{3.23}$$

$$\tilde{H}^{(t)} = \begin{bmatrix} \alpha^{(1)} & \beta^{(1)} & 0 & \cdots & 0 \\ \beta^{(1)} & \alpha^{(2)} & \beta^{(2)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \beta^{(t-1)} & \alpha^{(t)} \\ 0 & \cdots & 0 & 0 & \beta^{(t)} \end{bmatrix} \tag{3.24}$$

In order to solve the system of linear equations of Equation 3.18, it is used the MINRES algorithm [35]. At each step t, it is approximated the solution by the vector in the Krylov subspace at iteration t that minimizes the norm of $r^{(t)} = b - \Delta_{uu} Y_u^{(t)}$. As $Y_u^{(t)}$ is in the Krylov subspace, it can be rewritten as a linear combination of the columns of the Krylov matrix $Kr^{(t)}$, like in Equation 3.25, so now the objective is to find the vector c that reduces this expression.

$$\arg\min_{c \in \mathbb{R}^t} \|\Delta_{uu} Kr^{(t)} c - b\| \tag{3.25}$$

Since $Q^{(t)}$ spans the same space as $Kr^{(t)}$, we use a linear combination of the columns of $Q^{(t)}$, so the least square problem now is as the one in Equation 3.26.

$$\arg\min_{c \in \mathbb{R}^t} \|\Delta_{uu} Q^{(t)} c - b\| \tag{3.26}$$

And from Equation 3.23, Equation 3.26 can be rewritten as Equation 3.27, So now the problem is just of $(t+1) \times t$ dimensions.

$$\arg\min_{c \in \mathbb{R}^t} \|Q^{(t+1)} \tilde{H}^{(t)} c - b\| \tag{3.27}$$

$Q^{(t)}$ is orthonormal, so Equation 3.27 can be rewritten as Equation 3.28.

$$\arg\min_{c \in \mathbb{R}^t} \|\tilde{H}^{(t)} c - {Q^{(t+1)}}^\top b\| \tag{3.28}$$

The algorithm starts the iterations with $\beta^{(0)} = 0$, $q^{(0)} = 0$ and $q^{(1)} = b/\|b\|$, then ${Q^{(t+1)}}^\top b = \|b\| e_1$, where e_1 is a unit vector with one in its first entry. So we have a least-squares problem at each iteration t as shown in Equation 3.29.

$$\arg\min_{c \in \mathbb{R}^t} \|\tilde{H}^{(t)} c - \|b\| e_1\| \tag{3.29}$$

Finally the current estimation t of Y_u is given by $Y_u^{(t)} = Q^{(t)} c$. Summarizing the MINRES algorithm at each iteration $t = 1, 2, 3...$ calculates the parameters of Equation 3.29 ($\alpha^{(t)}$ and $\beta^{(t)}$ for $\tilde{H}^{(t)}$) and also $Q^{(t)}$, solves a least square problem of size $(t+1) \times t$ and then calculates $Y_u^{(t)}$,

as is shown in Algorithm 1.

Algorithm 1 MINRES Algorithm for Krylov Subspace Iteration

Require: $it \in \mathbb{N}, \Delta, W, Y_l$
  $\beta^{(0)} \Leftarrow 0,\ q^{(0)} \Leftarrow 0,\ q^{(1)} \Leftarrow b/\|b\|$
  $t \Leftarrow 1$
  while $t \leq it$ **do**
    $v \Leftarrow \Delta_{uu} q^{(t)}$
    $\alpha^{(t)} \Leftarrow {q^{(t)}}^{\top} v$
    $v \Leftarrow v - \beta^{(t-1)} q^{(t-1)} - \alpha^{(t)} q^{(t)}$
    $\beta^{(t)} \Leftarrow \|v\|$
    $q^{(t+1)} \Leftarrow v/\beta^{(t)}$
    $c \Leftarrow leastSquareOptimization(\|\tilde{H}^{(t)} c - \|b\| e_1\|)$
    $Y_u^{(t)} \Leftarrow Q^{(t)} c$
    $t \Leftarrow t + 1$
  end while
  $return\ Y_u^{(t)}$

3.4.2 Manifold Learning using Isomap and Nystrom approximate spectral decomposition

This method uses a graph in order to discover the topological relations of the input data and maps them into geodesic distances, which later would be treated as euclidean distances, by applying a KNN classifier in the transformed space, and by this, propagate the labels from the labeled data to the unlabeled data. This technique of non linear dimensionality reduction is called Isomap or Isometric Feature Mapping [45, 49]. The idea behind the algorithm is to use the Isomap reduction to transform the data and then apply a traditional KNN classifier; so each unlabeled vertex would have the label of the labeled vertex that has the shortest path to it.

The Isomap method has three steps. First we build an undirected neighborhood graph from all the labeled and unlabeled data-points. This step follows the classical semi-supervised learning building graph procedures, but instead of similarity as the edge weights it would be just the distance between the two vertices as the edge weight. Also this graph has to be an ϵ-graph or a k-graph. The second step computes a $(l+u) \times (l+u)$ matrix of geodesic distances D_G for all pair of nodes. By geodesic distance we refer to the shortest paths from one vertex to all

the other vertices. Then the matrix D_G is centered as the procedure in Equations 3.30 and 3.31 shows, where I_{ij} is the identity matrix of size $(l+u) \times (l+u)$.

$$\tau(D_G) = -HSH/2 \tag{3.30}$$

$$\begin{aligned} S_{ij} &= D_{G_{ij}}^2 \\ H_{ij} &= I_{ij} - \tfrac{1}{(l+u)} \end{aligned} \tag{3.31}$$

Finally the last step of Isomap takes the centered matrix $\tau(D_G)$ and gets its spectral decomposition. By this the new low-dimensional data representation X' (of p number of dimensions), would be given by Equation 3.32, where Σ^p is the diagonal matrix storing the top p eigenvalues of $\tau(D_G)$ ($\Sigma^{p\,1/2}_{ij} = \sqrt{\Sigma^p_{ij}}$ for $0 < i, j < l+u+1$), and U^p are the associated eigenvectors ($\tau(D_G)U^p_i = U^p_i \Sigma^p_{ii}$ for $0 < i < l+u+1$).

$$X' = \Sigma^{p\,1/2} U^{p\top} \tag{3.32}$$

In order to speed up the Isomap reduction method, it is necessary to apply an approximate spectral decomposition in its third step, where the $\tau(D_G)$ matrix can be very large and expensive to decompose in its eigenvectors and eigenvalues. The Nystrom method can calculate approximate spectral decomposition of a matrix in an acceptable computational cost [2, 15, 30]. If the $\tau(D_G)$ matrix is randomly sampled in $r \ll l+u$ columns, then we have the matrix $\tau(D_G)_s$ of $l+u \times r$ size. As $\tau(D_G)$ is a symmetric positive semi-definite matrix it can be rearranged as in Equation 3.33, where M are the intersections of those r random sampled columns.

$$\tau(D_G) = \begin{bmatrix} M & S_{21}^\top \\ S_{21} & S_{22} \end{bmatrix} \qquad \tau(D_G)_s = \begin{bmatrix} M \\ S_{21} \end{bmatrix} \tag{3.33}$$

From $\tau(D_G)_s$, the eigenvalues (Σ_M) and eigenvectors (U_M) of matrix M, can approximate the eigenvalues and eigenvectors of $\tau(D_G)$ as Equations 3.34 and 3.35 shows. $\tilde{\Sigma}$ and $\tilde{U}$ are the approximated eigenvalues and eigenvectors of $\tau(D_G)$ respectively, and Σ_M^+ is the pseudo inverse of Σ_M.

$$\tilde{\Sigma} = \left(\frac{l+u}{r}\right) \Sigma_M \tag{3.34}$$

$$\tilde{U} = \sqrt{\frac{r}{l+u}} \tau(D_G)_s U_M \Sigma_M^+ \tag{3.35}$$

The Isomap method is an non-linear dimensional reduction method, which combined with the KNN classifier in the transformed space, accomplishes a transductive learning task using graph. And sped up with the Nystrom approximation it can be seen as a graph based SSL algorithm for fast computation. In our Transductive Graph based SSL framework (see Figure 3.2), this method has a different graph construction approach, since it has to use distances instead of similarities (for closest paths calculations) and can only use sparse graphs, so then the Graph regularization would start in the second step of the Isomap process (see Algorithm 2).

Algorithm 2 Isomap with Nystrom Algorithm

Require: W, Y_l, p, r "W represents a sparse graph"
$D_G \Leftarrow geodesicDistances(W)$
$\tau(D_G) \Leftarrow centerMatrix(D_G)$
$(\tilde{\Sigma}, \tilde{U}) \Leftarrow nystromApproximation(\tau(D_G), r)$
$(\Sigma^p, U^p) \Leftarrow selectTop(\tilde{\Sigma}, \tilde{U}, p)$ "Select highest eigenvalues and its corresponding eigenvectors"
$X' \Leftarrow \Sigma^{p1/2} U^{p\top}$
$Y_u = KNN(X', Y_l)$
$return\ Y_u$

3.4.3 Eigenvectors from the Smoothness Operator of the Graph

In this method it is used as regularization scheme the expression of Equation 3.36 [14, 40], where λ is a hyper-parameter of the algorithm.

$$\underset{f \in \mathbb{R}^{l+u}}{\arg\min} \sum_{i \in X_l} \lambda (f(x_i) - y_i)^2 + \frac{1}{2} \sum_{i,j \in X} w_{ij} (f(x_i) - f(x_j))^2 \tag{3.36}$$

In matrix notation Equation 3.36 can be rewritten as Equation 3.37, where Λ is a diagonal matrix where $\Lambda_{ii} = \lambda$ for labeled data-points and $\Lambda_{ii} = 0$ for unlabeled data-points.

$$\underset{f \in \mathbb{R}^{l+u}}{\arg\min} (f - Y_l)^\top \Lambda (f - Y_l) + f^\top \Delta f \tag{3.37}$$

The closed form solution for Equation 3.37 is in Equation 3.38. This solution requires solving an $(l+u) \times (l+u)$ system of linear equations. So, for large $(l+u)$, it can be prohibitive to solve it, because it implies a heavy matrix inversion. But the dimension of the problem can be reduced by just working with a small number of eigenvectors of the graph Laplacian.

$$(\Delta + \Lambda)f = \Lambda Y_l \tag{3.38}$$

Note that the regularizer in the regularization scheme ($f^T \Delta f$) denotes smoothness over the graph label transitions. Within it, any possible f can be approximated as a linear combination of the eigenvectors ϕ of the graph Laplacian ($f = \sum_i \alpha_i \phi_i$, where $\Delta\phi_i = \sigma_i \phi_i$) with the smallest eigenvalues σ, because $smoothness(\phi_i) = \phi_i^\top \Delta \phi_i = \sigma_i$. As the smoothness of f has to be minimized, the smoothness of any f would be given by $\sum_i \alpha_i \sigma_i$. By this way $f = U\alpha$, where U is a $l + u \times p$ matrix whose columns are the p eigenvectors of smallest eigenvalues. Using the eigenvectors of the smoothness operator of the graph (the graph Laplacian) with smallest eigenvalues, the solution to Equation 3.38 can be approximated by Equation 3.39, where we just have to find the vector α of p size.

$$(U^\top \Delta U + U^\top \Lambda U)\alpha = U^\top \Lambda Y_l \tag{3.39}$$

This algorithm makes calculations with a smaller matrix, built up from the graph Laplacian. It is shown in Algorithm 3.

Algorithm 3 Eigenvectors from the Smoothness Operator Algorithm

Require: Y_l, Δ, p
$U \Leftarrow smallestEigenvalues(\Delta, p)$
$\alpha \Leftarrow solveSystem(U^\top \Delta U + U^\top \Lambda U, U^\top \Lambda Y_l)$
$Y = U\alpha$
$return\ Y$

3.4.4 Eigenfunctions from the Graph Laplacian

This method uses the same minimization expression as the previous method (Equation 3.36 or 3.37). The idea is to approximate the eigenvectors of the graph Laplacian by interpolating the eigenfunctions of a weighted smoothness operator Δ_p [14]. So that, when $l + u \to \infty$, the smoothness regularizer ($(1/(l+u)^2) f^T \Delta f$) will approach Δ_p. Δ_p is defined by the assumed densities of the data $p(x)$, and works for any function $F(x)$ defined on $\mathbb{R}^d$ (d is the dimension of the data-points) as shown in Equation 3.40, $W(x_1, x_2) = exp(-\|x_1 - x_2\|^2/2\epsilon^2)$.

$$\Delta_p(F) = \frac{1}{2} \int (F(x_1) - F(x_2))^2 W(x_1, x_2) p(x_1) p(x_2) dx_1 x_2 \tag{3.40}$$

As the smoothness regularizer has to be minimized so is the weighted smoothness operator Δ_p, and any eigenvalue σ_p of any given eigenfunction ϕ_p is its smoothness $\sigma_p = \Delta_p(\phi_p)$. We are working with a set of discrete points, from which we can calculate a discrete density, and then calculate the eigenfunctions numerically with the Equation 3.41, where $\tilde{W}$ is the affinity between the discrete data-points, P is a diagonal matrix whose elements are the densities of the data-points, $\tilde{D}$ is another diagonal matrix whose elements are the sum of the columns of $P\tilde{W}P$, $\hat{D}$ is a diagonal matrix whose elements are the sum of the columns of $P\tilde{W}$, g are the eigenfunctions and σ the corresponding eigenvalues.

$$(\tilde{D} - P\tilde{W}P)g = \sigma P\hat{D}g \tag{3.41}$$

The eigenfunctions of a set of discrete data-points can be found firstly by finding rotations for the data that make them as independent/uncorrelated as possible, for example using principal component analysis (PCA) or independent component analysis (ICA) over the data and working with each component one by one. Then a histogram has to be approximated for each independent/uncorrelated component. After that, solve Equation 3.41, this can be done by solving an eigenvalue problem for a $B \times B$ matrix, where B is the number of the bins of the histogram. Finally sort the eigenfunctions by their increasing eigenvalues and choose the p-first eigenfunctions, then interpolate linearly the data by each chosen eigenfunctions with the corresponding bins generating approximated eigenvectors. At the end use the approximated eigenvectors in Equation 3.39, and find the solution to the linear system.

Algorithm 4 Eigenfunctions from the Graph Laplacian Algorithm

Require: X, Y_l, p, B
$X' \Leftarrow independentRotations(X)$
$H \Leftarrow calculateHistogram(X')$
$\tilde{W}, P \Leftarrow get\tilde{W}AndP(H)$
$\tilde{D} \Leftarrow getColumnsSum(P\tilde{W}P)$
$\hat{D} \Leftarrow getColumnsSum(P\tilde{W})$
$G, \Gamma' \Leftarrow getEigenfunctions(\tilde{D} - P\tilde{W}P, P\hat{D})$ "G is a matrix which columns are the numerical eigenfunctions"
$U'' \Leftarrow interpolate(X', G)$
$U', \Gamma \Leftarrow selectEigenvectors(U'', \Gamma', p)$
$\alpha \Leftarrow solveSystem(\Gamma + U'^{\top}\Lambda U', U'^{\top}\Lambda Y_l)$
$Y = U\alpha$
$return\ Y$

This method is particularly interesting because it does not need to calculate the graph Laplacian, moreover you do not need to build the graph, because we approximate the eigenvectors of the graph Laplacian by interpolating them from the numerically calculated eigenfunctions, which were calculated from the data histogram, so there is no need to build the graph, we just need the data. Then, as we are assuming that $U'^{\top}\Delta U' = \Gamma$, where Γ is a diagonal matrix with the p smallest eigenvalues (smoothness operator) and U' are the approximated eigenvectors, Equation 3.39 would be as Equation 3.42, where all the parameters are independent from the graph similarity matrix W and graph Laplacian Δ. Algorithm 4 shows the complete method.

$$(\Gamma + U'^{\top}\Lambda U')\alpha = U'^{\top}\Lambda Y \tag{3.42}$$

Chapter 4

GNG Graph Construction approach

In this chapter, we describe our first contribution. It is a graph construction algorithm based in an unsupervised neural network called Growing Neural Gas (GNG). This new graph construction approach aims to represent the data set with fewer vertices than data-points in order to save memory for later steps in the semi-supervised learning with graph process. The resulting graph would be call as GNG-Graph. First we are going to describe the traditional Growing Neural Gas from the literature; after that, we introduce a new index that describes the state of a graph with respect to the data set it is mapping; and finally we explain how this index is used within the GNG algorithm as a stopping criterion, and how it works in the semi-supervised learning scheme.

4.1 Growing Neural Gas Algorithm

A GNG neural network is an unsupervised incremental algorithm that learns topological relations of a given input data set. This neural network can change its configuration while it is training itself, adding and removing neurons and connections whenever necessary [16, 9, 18, 43]. Unlike other unsupervised neural networks, this behavior optimizes the speed and memory use.

This neural network can be seen as a graph $G = (V, E)$, where each vertex $v \in V$ is a neuron and the edges $e \in E$ are the connections between the neurons. Each vertex has the following structure: $v = (p, \xi)$, where p are the coordinates of the neuron v, $p \in \mathbb{R}^n$ (as the input data-points $x \in \mathbb{R}^n$) and $\xi \in \mathbb{R}$ is the associated error of neuron v. Each vertex should represent a subset of data-points with high similarity, so $|V| \leq (l + u)$.

The edges has also a configuration structure; each edge $e = (v_i, v_j, w_{ij}, t_{ij})$, v_i and v_j are the adjacent vertices to e, w_{ij} is the edge's weight (which could be the euclidean distance between them) and t_{ij} is the edge's age, in terms of training iterations. The classic GNG algorithm is

explained below:

- The GNG begins with just two isolated neurons ($|V| = 2$ and $|E| = 0$), randomly disposed in the data-points' space.
- At each iteration one data-point x is analyzed (we call as iteration every time an input pattern is analyzed). The two nearest neighbors to x are picked from V, say v_s and v_t, being v_s the closest vertex to x. Every time a vertex v_s is found to any data point x, we will say that the neuron v_s was activated by x.
- Then the error rate ξ_s of v_s is updated as in Equation 4.1; where $d(x, p_s)$ (d is a distance function, ex: Euclidean distance) is the distance between x and v_s (which is at p_s coordinates), by this way the vertices that were activated by more data-points tend to have a bigger error ξ.

$$\xi_s = \xi_s + d(x, p_s) \tag{4.1}$$

- After updating the error, we create an edge $e_{st} = (v_s, v_t, 0, 0)$, if it already exists then just the age t_{st} is set to 0.
- Then the coordinates p_s of v_s and the coordinates p_m of all of its adjacent vertices including v_t are updated as in Equations 4.2 and 4.3. ϵ_s and ϵ_m are real numbers in $[0, 1]$, and $\epsilon_s \geq \epsilon_m$, those are hyper parameters of the GNG.

$$p_s = p_s + \epsilon_s(x - p_s) \tag{4.2}$$

$$p_m = p_m + \epsilon_m(x - p_m) \tag{4.3}$$

- The distance between v_s and all of its adjacent vertices v_m are updated, so all the corresponding edges e_{sm} would be $e_{sm} = (v_s, v_m, d(p_s, p_m), t_{sm})$.
- At each iteration all GNG's edges connected to v_s (e_{sm}) increment their age t_{sm} by one, so now $e_{sm} = (v_s, v_m, w_{sm}, t_{sm} + 1)$.

- The neural network grows when a given number λ of input patterns x are analyzed, λ is also a hyper-parameter of this algorithm.

- Every λ iterations happens it is selected the vertex v_{max} with maximum internal error ξ_{max}, and then it is selected the adjacent vertex v_{nmax} to v_{max} that has the maximum error from all of v_{max}'s adjacent vertices.

- It is created a new vertex v_{new} between v_{max} and v_{nmax}, $p_{new} = 0.5(p_{max} + p_{nmax})$. Subsequently the errors of v_{max} and v_{nmax} are decreased as in Equation 4.4, after this the error rate of the new neuron is updated as in Equation 4.5, α is another hyper-parameter of the algorithm.

$$\begin{aligned} \xi_{max} &= \xi_{max} - \alpha(\xi_{max}) \\ \xi_{nmax} &= \xi_{nmax} - \alpha(\xi_{nmax}) \end{aligned} \tag{4.4}$$

$$\xi_{new} = 0.5(\xi_{max} + \xi_{nmax}) \tag{4.5}$$

- This algorithm also prunes vertices and edges from the graph. There is a hyper-parameter β; at each iteration all the edges with ages $t > \beta$ are pruned and if one vertex after this becomes isolated it is removed too. Finally all the errors of all the remaining vertices are decreased by $\xi = \xi - \delta\xi$, δ is the last hyper-parameter of this algorithm. The classic GNG algorithm is specified in Algorithm 5.

4.2 Graph Mapping Index

Using a correct stopping criterion for our problem, that represents correctly at the right moment (in the training process) the built graph with respect to the group of data-points is a crucial step in our graph construction algorithm. The main idea of this algorithm is to use the GNG with some stopping criteria to build a graph that faithfully represents $X_l \cup X_u$, and immediately after this use some traditional graph regularization algorithm in order to calculate the classification function f over the graph G built by the GNG algorithm. In order to know when the built graph is ready we introduce a graph construction index.

The index that will be used to stop the GNG training has to tell us if there are significant changes in the mapping of the data by the graph at every iteration. We call this index as Graph

Algorithm 5 Classic GNG

Require: P "Input patterns"
Require: $\epsilon_s, \epsilon_m, \beta, \lambda, \alpha, \delta$
Require: it "Maximum number of times every pattern is analyzed"
  $i \Leftarrow 1,\ A.initialize(2)$ "A is the set of GNG's neurons and connections"
  while $i \leq it$ **do**
    $j \Leftarrow 0$
    while $j < Size(P)$ **do**
      $v_s \Leftarrow Nearest(A, P[j])$
      $v_t \Leftarrow Nearest(A - v_s, P[j])$
      $IncrementEdgeAges(v_s)$
      $UpdateError(v_s, P[j])$
      $UpdateVector(v_s, \epsilon_s)$
      $UpdateVector(Neighbors(v_s), \epsilon_m)$
      if $AreConnected(v_s, v_t)$ **then**
        $SetAge(Edge(v_s, v_t), 0)$
      else
        $Connect(v_s, v_t)$
      end if
      $RemoveEdges(A, \beta)$
      if $Module(j, \lambda) = 0$ **then**
        $v_{max} \Leftarrow MaxError(A)$
        $v_{nmax} \Leftarrow MaxError(Neighbors(v_{max}))$
        $v_{new} \Leftarrow InsertBetween(v_{max}, v_{nmax})$
        $Disconnect(v_{max}, v_{nmax})$
        $Connect(v_{max}, v_{new})$
        $Connect(v_{nmax}, v_{new})$
        $DecreaseError(v_{max}, \alpha)$
        $DecreaseError(v_{nmax}, \alpha)$
        $SetError(v_{new}, 0.5(getError(v_{max}) + getError(v_{nmax})))$
      end if
      $DecreaseError(A, \delta)$
      $j \Leftarrow j + 1$
    end while
    $i \Leftarrow i + 1$
  end while
  $return\ A$

Mapping Index (GMI). The GMI is composed by two other sub-indices.

First we have the *Vertex Influence Index* I_{vi}, see Equation 4.6; p_{x_i} are the coordinates of the neuron that was the last to be activated by x_i, $x_i \in X_l \cup X_u$ ($|X_l| = l$ and $|X_u| = u$). When each pattern x_i is analyzed by the GNG, it is annotated the vertex that was activated by it. This index is a mean of all the distances of each neuron to their respective activation data-points. It tells us basically if there are enough vertices that map the data. One neuron records only the last data-point that activated it, also depending of the topology of the data set and the initial hyper parameters, one data-point can activate more than one neuron.

This index has to be minimized for better results, but also depending of the topology of the input data set, this index would reach a minimum that cannot be decreased by any other graph configuration.

$$I_{vi} = \frac{1}{|V|} \sum_{i=1}^{|V|} d(p_{x_i}, x_i) \tag{4.6}$$

The second index we use is the *Vertex Distribution Index* I_{vd}, this index is shown in Equation 4.7. $Adj(v_i)$ is a function that returns all the adjacent vertices to the vertex v_i, p_i and p_b are the coordinates of vertices v_i and v_b, the distance $d(p_i, p_b)$ between them, are obtained from the weight w_{ib} of the edge with v_i and v_b as adjacent vertices; finally $|V|$ represents the number of vertices in V. The vertex distribution index is the mean of the larger edge incident to each vertex in the graph, it express if the graph is faithfully distributed. This index for better results in this problem should have the least possible value. I_{vd} could reach a very low value if the graph has too many vertices. From this it has to work with the vertex influence index for a good description of the data graph mapping.

$$I_{vd} = \frac{1}{|V|} \sum_{i=1}^{|V|} \max_{v_b \in Adj(v_i)} d(p_i, p_b) \tag{4.7}$$

When a graph faithfully maps the input data, $I_{vi} \leq I_{vd}$, because we are assuming that the vertices are uniformly distributed and the original data-points are almost at the same distance to their activated neuron. The mean of all the distances of the data-points to their respective activated neuron can not be larger than the distance between vertices. From this we get to the final graph mapping index I_{gm} in Equation 4.8, which should be maximized, but never greater than one. This index tell us the current state of the graph with respect to the data, every time the input patterns are analyzed by the GNG.

$$I_{gm} = \frac{I_{vi}}{I_{vd}} \tag{4.8}$$

4.3 Stopping Criterion

As the literature points out, in order to train a "good" GNG it is necessary to use a stopping criteria, different than the number of iterations [9]. This way we optimize for time and space, avoiding unnecessary training iterations, vertices and edges.

The GNG training with stopping criteria would be done in the same fashion as was done by D. Chavez et al. [9] using an index error and an error factor, but replacing the SV index by the Graph Mapping index, which tells us the state of the data graph mapping at every configuration of the graph. Each time that a quantity of input patterns are analyzed (we set this quantity as the size of the input-patterns set $l + u$), it is calculated the I_{gm} index for all the analyzed patterns ($I_{gm} = 0$ at the beginning). Then it is calculated the index error e_I of the I_{gm} index, which is the difference between the actual I_{gm} and the last I_{gm}. The error factor e_f is the number of times that the index error can be less than the accepted error e_a (the accepted error is an hyper-parameter of the algorithm) before stopping the algorithm.
Also another modification was made to the traditional GNG. To insert a new vertex to the graph, not only, it has to satisfy the condition of the λ patterns analyzed, but also there has to be checked if there are no more vertices than a proportion p_r of the input patterns, this proportion p_r is also an hyper-parameter of the algorithm.
The last modification done to the traditional GNG was that if, after the training process was done, it is obtained a not connected graph, this graph is converted to a connected graph by connecting the vertices of different subgraphs that are the closest between them.

After the successful training of the GNG with the graph mapping index as the stopping criteria, we have a graph where its edge weights represent the distance between vertices, so now we have to convert this distances into similarities (see section 3.2.4), after this the traditional graph regularization would be done over the graph; but just before this we have to deal also with the labels mapping. Basically a vertex would have the label that is most common for all the labeled data-points that activated it, if those data-points do not have a label then the vertex either does not have a label, or if there are more than one common label it would be chosen randomly. After the regularization step, the label mapping is inverted, all the data-points would have the label of the neuron that was activated by them, but all the data-points in X_l would

Algorithm 6 GNG Graph Construction

Require: P "Input patterns"
Require: $\epsilon_s, \epsilon_m, \beta, \lambda, \alpha, \delta$
Require: it "Maximum number of times every pattern is analyzed"
Require: p_r, e_a, e_f
  $i \Leftarrow 1,\ c \Leftarrow 0,\ I_{gm}^{(0)} \Leftarrow 0,\ A.initialize(2)$ "A is the set of GNG's neurons and connections"
  while $i \leq it$ **do**
    $j \Leftarrow 0$
    while $j < Size(P)$ **do**
      $v_s \Leftarrow Nearest(A, P[j])$
      $v_t \Leftarrow Nearest(A - v_s, P[j])$
      $IncrementEdgeAges(v_s)$
      $UpdateError(v_s, P[j])$
      $UpdateVector(v_s, \epsilon_s)$
      $UpdateVector(Neighbors(v_s), \epsilon_m)$
      if $AreConnected(v_s, v_t)$ **then**
        $SetAge(Edge(v_s, v_t), 0)$
      else
        $Connect(v_s, v_t)$
      end if
      $RemoveEdges(A, \beta)$
      if $Module(j, \lambda) = 0\ \wedge\ |A| < p_r|P|$ **then**
        $v_{max} \Leftarrow MaxError(A)$
        $v_{nmax} \Leftarrow MaxError(Neighbors(v_{max}))$
        $v_{new} \Leftarrow InsertBetween(v_{max}, v_{nmax})$
        $Disconnect(v_{max}, v_{nmax})$
        $Connect(v_{max}, v_{new})$
        $Connect(v_{nmax}, v_{new})$
        $DecreaseError(v_{max}, \alpha)$
        $DecreaseError(v_{nmax}, \alpha)$
        $SetError(v_{new}, 0.5(getError(v_{max}) + getError(v_{nmax})))$
      end if
      $DecreaseError(A, \delta)$
      $j \Leftarrow j + 1$
    end while
    $I_{gm}^{(i)} \Leftarrow GraphMappingIndex()$
    $e_I \Leftarrow I_{gm}^{(i)} - I_{gm}^{(i-1)}$
    if $e_I \leq e_a$ **then**
      $c \Leftarrow c + 1$
      if $c \geq e_f$ **then**
        $return\ A$
      end if
    else
      $c \Leftarrow 0$
    end if
    $i \Leftarrow i + 1$
  end while

keep its original label. The GNG graph construction algorithm is shown in Algorithm 6, and the procedure to use a regularizer with the GNG-Graph is shown in Algorithm 7.

Algorithm 7 GNG Regularizer application

Require: Y_l
Require: P "Input patterns"
Require: R "Regularizer"
Require: Θ_{GNG}, Θ_R "GNG's hyper parameters and Regularizer's hyper parameters"
$G \Leftarrow GNGGraphConstructionAlgorithm(P, \Theta_{GNG})$
if $Gisnotconnected$ **then**
$G \Leftarrow ConnectGraph(G)$
end if
$G \Leftarrow ConvertDistancesToSimilarities(G)$
$G \Leftarrow LabelsMapping(G, P, Y_l)$
$Y_g \Leftarrow ApplyRegularizer(R, G, Y_l)$
$Y_u \Leftarrow InvertLabelsMapping(P, G, Y_g)$
$return\ Y_u$

We present an example of the GNG's graph construction procedure below. Initially we have a data set ($|X| = 29$), from which we are going to build the graph (see Figure 4.1). Each data-point in this data set belongs to one of two classes, and we just know one labeled data-point from each class ($|X_l| = 2$ and $|X_u| = 27$).

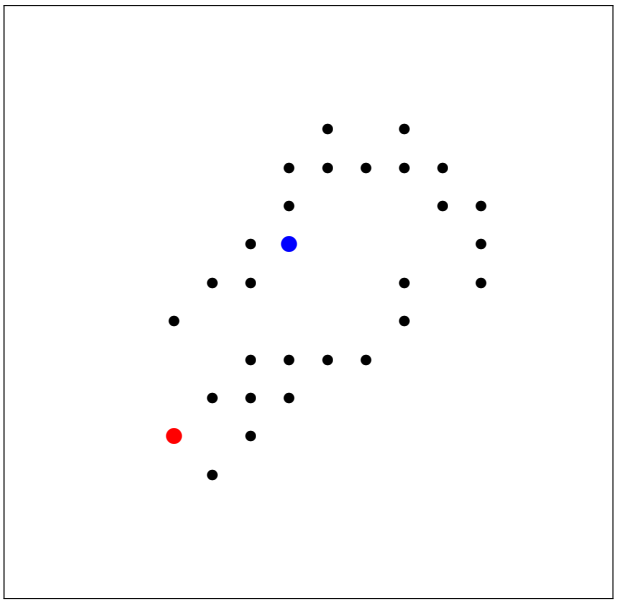


Figure 4.1: A random data set, with two labeled data-points ($|X| = 29$, $|X_u| = 27$, $|X_l| = 2$).

The GNG will initialize with two isolated vertices with random coordinates (see Figure 4.2).

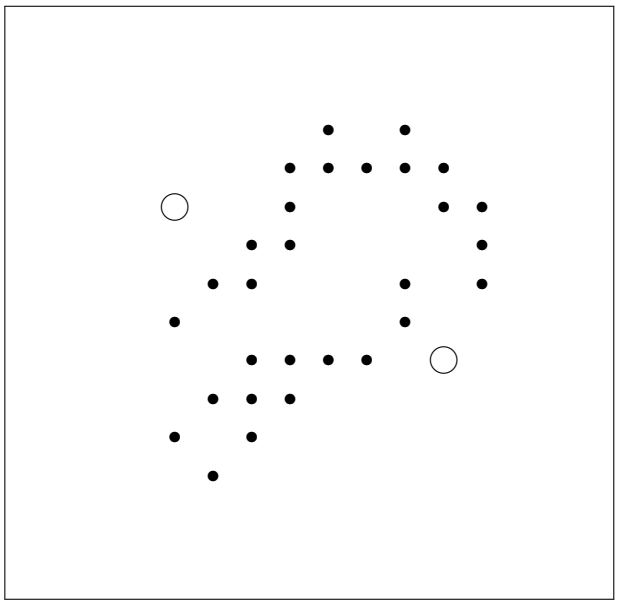


Figure 4.2: GNG's initialization, two vertices with random coordinates ($|V| = 2$, $|E| = 0$).

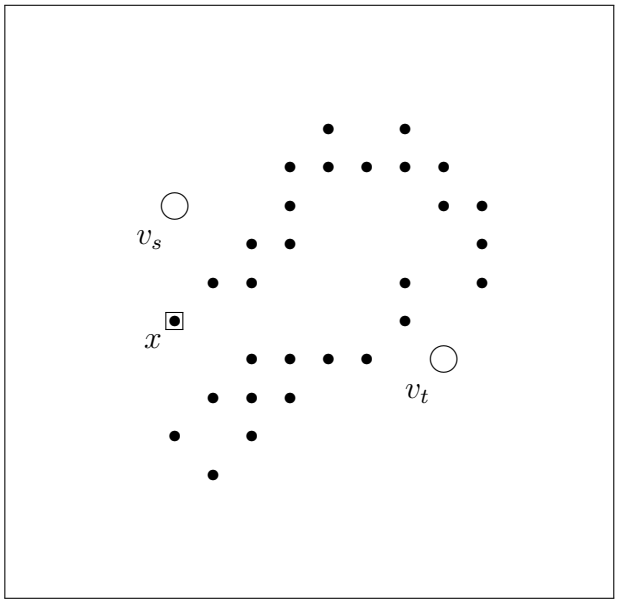


Figure 4.3: GNG, analyzing one data-point x, finding v_s and v_t.

Then at each iteration is analyzed one data-point x, the GNG finds the closest neuron v_s to

x and the second closest neuron v_t (see Figure 4.3). Then the internal error ξ_s of vertex v_s is updated as in Equation 4.1. After that the GNG creates a edge between v_s and v_t, as Figure 4.4 shows.

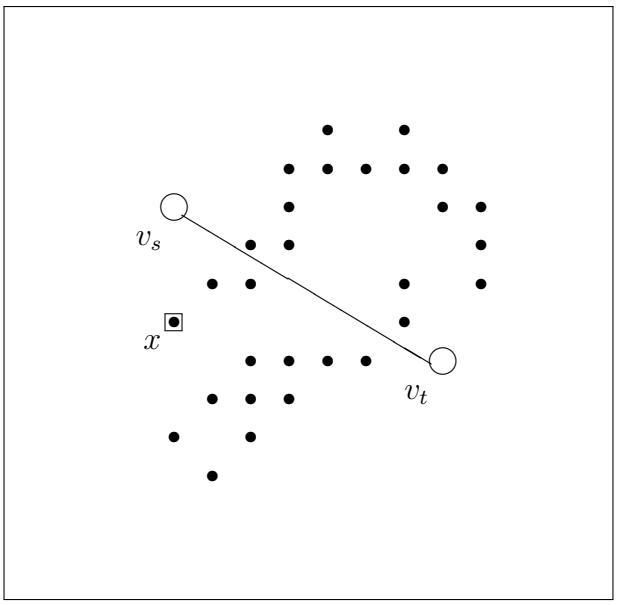


Figure 4.4: GNG, connecting v_s and v_t.

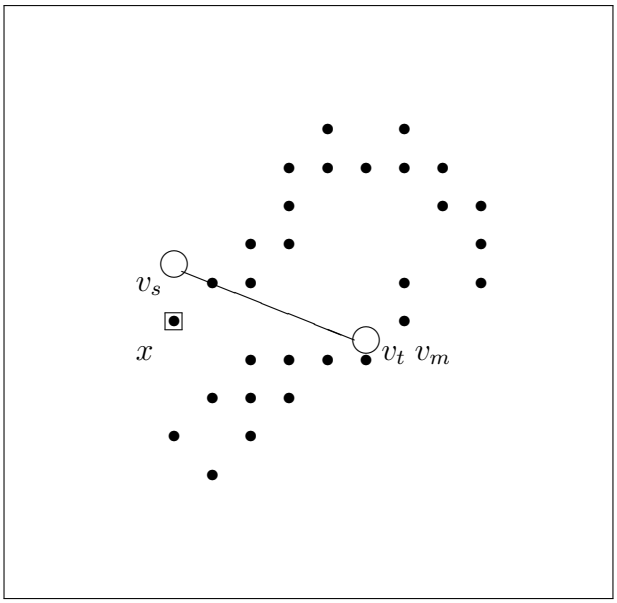


Figure 4.5: GNG, updating the coordinates p_s (v_s) and p_m (v_m), v_t is also a v_m vertex.

The next stage, the GNG using equations 4.2 and 4.3 modifies the coordinates p_s of v_s and the coordinates p_m of all v_s's adjacent vertices v_m as shown in Figure 4.5, when $|V| = 2$ as in this case, the only v_m is v_t. After each data-point is analyzed, the distances of all the modified edges in the current iteration are updated and the edges' ages of all the adjacent edges to the current v_s are increased by one.

When λ data-points are analyzed (λ iterations) the GNG grows by adding one new vertex, firstly it finds the vertex with maximum internal error v_{max}, and v_{max}'s adjacent vertex with maximum internal error v_{nmax} (see Figure 4.6). Then it is inserted a new vertex v_{new} between v_{max} and v_{nmax} as Figure 4.7 shows. In this case when $|V| = 2$, one vertex will be v_{max} and the other will be v_{nmax}.

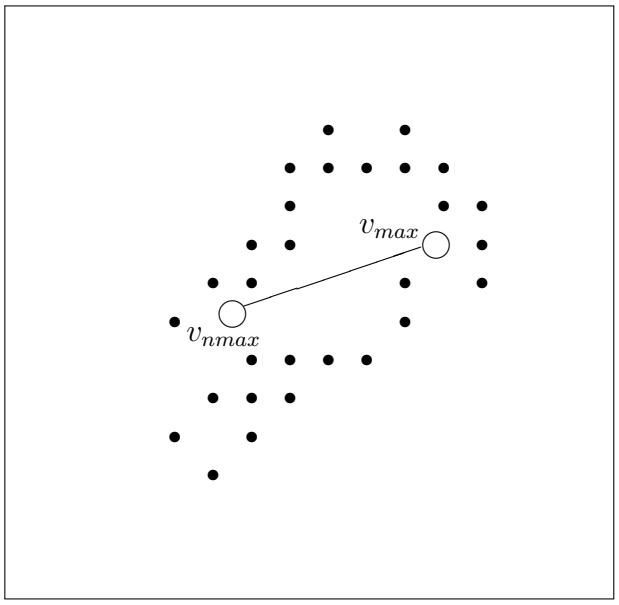


Figure 4.6: GNG, after λ iterations, it finds the vertex with maximum internal error v_{max}, and its corresponding adjacent vertex with maximum internal error v_{nmax}.

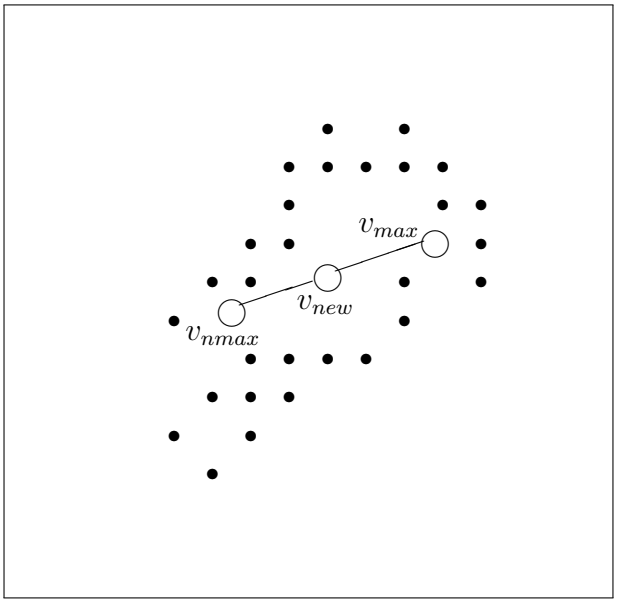


Figure 4.7: GNG, inserting a new vertex v_{new}.

After analyzing more data-points, the GNG would be as the one in Figure 4.8.

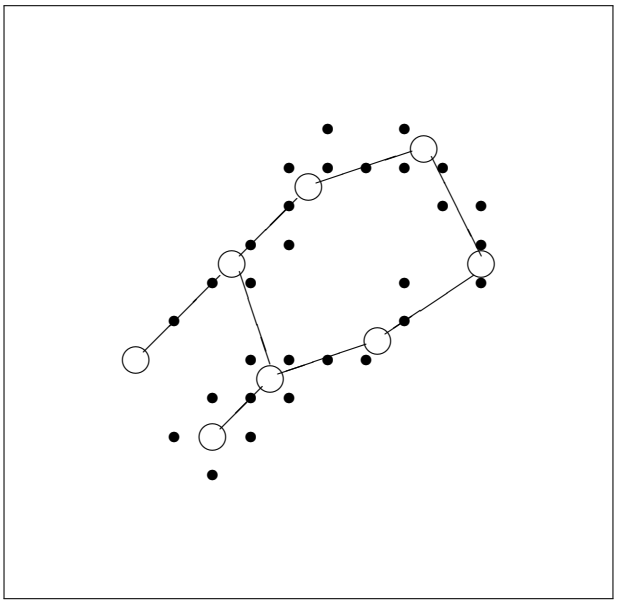


Figure 4.8: GNG in the middle of the training process, with many inserted vertices.

The final step of each iteration is removing all the edges with ages $t > \beta$ (see Figure 4.9) and right after that remove all isolated vertices (see Figure 4.10).

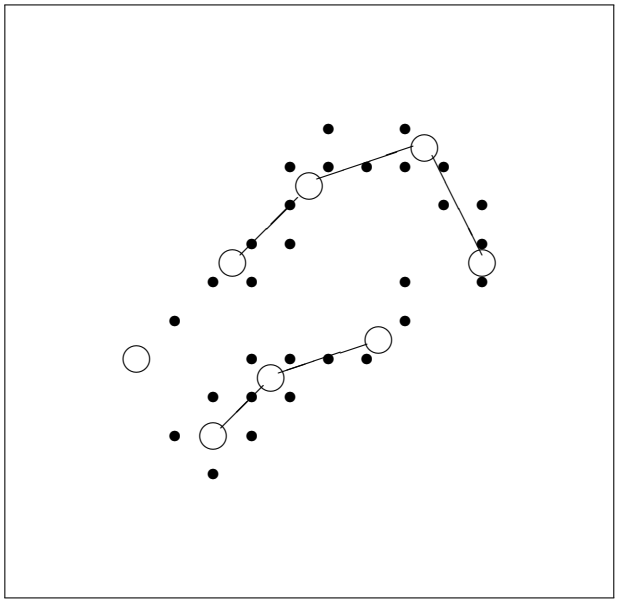


Figure 4.9: GNG, after removing the edges with age $t > \beta$ from Figure 4.8.

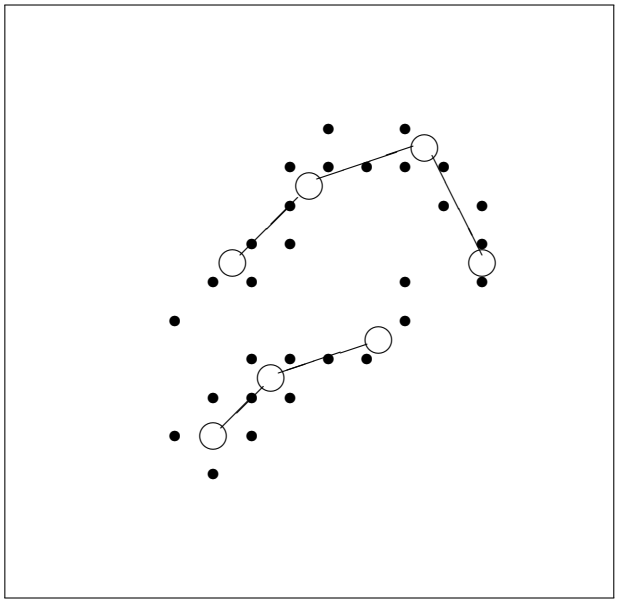


Figure 4.10: GNG, after removing the isolated vertices of Figure 4.9.

Figure 4.11 shows how the vertex influence is calculated (see Equation 4.6), depicting the vertex influence area of one vertex (in blue). Figure 4.12 shows how to calculate the vertex distribution (see Equation 4.7), depicting the larger edge of one vertex (in blue). Using the

vertex influence and the vertex distribution, the GNG calculates the graph mapping index (see Equation 4.8) each time that $l + u$ data-points are analyzed, this is shown in Figure 4.13.

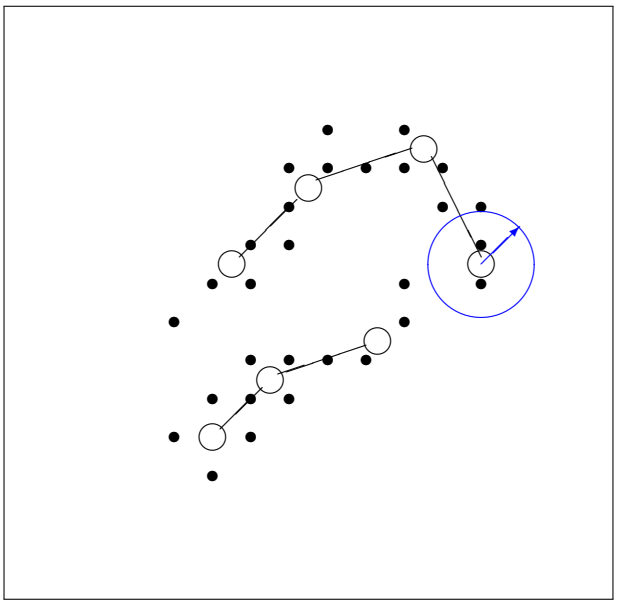


Figure 4.11: GNG, calculating the vertex influence.

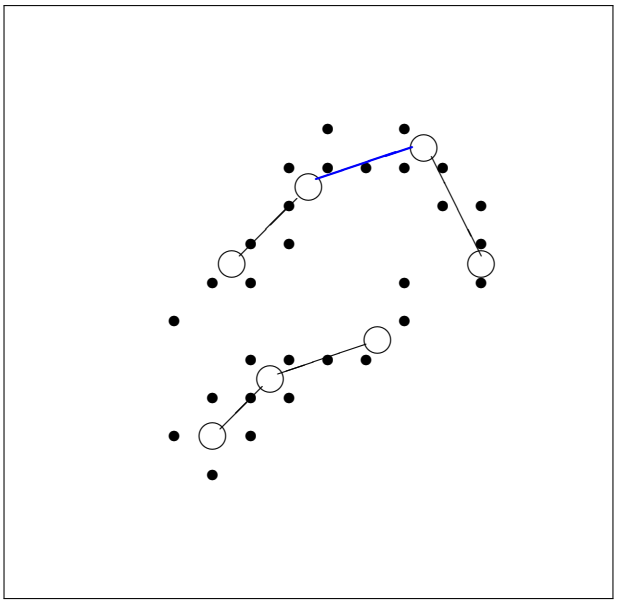


Figure 4.12: GNG, calculating the vertex distribution.

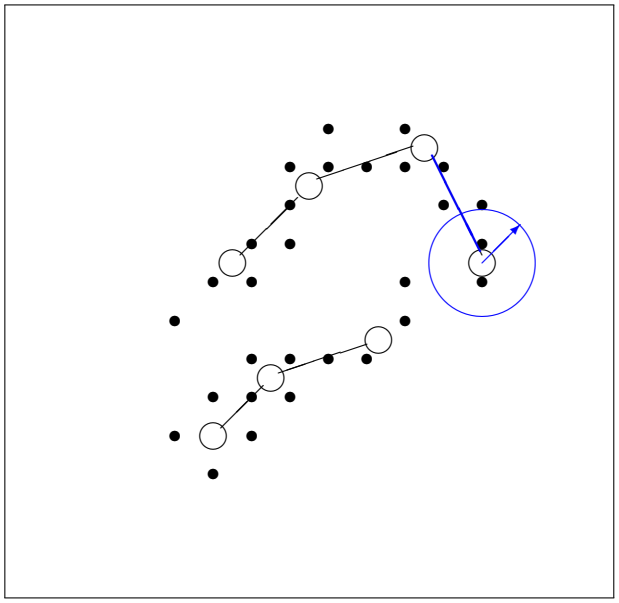


Figure 4.13: GNG, calculating the graph mapping index.

Finally the GNG will remain as in Figure 4.14, after mapping the labeled data-points' labels into the vertices. In this case the GNG has mapped 29 data-points into a graph of 7 vertices and 6 edges. And now a regularization method can be applied over this resulting graph.

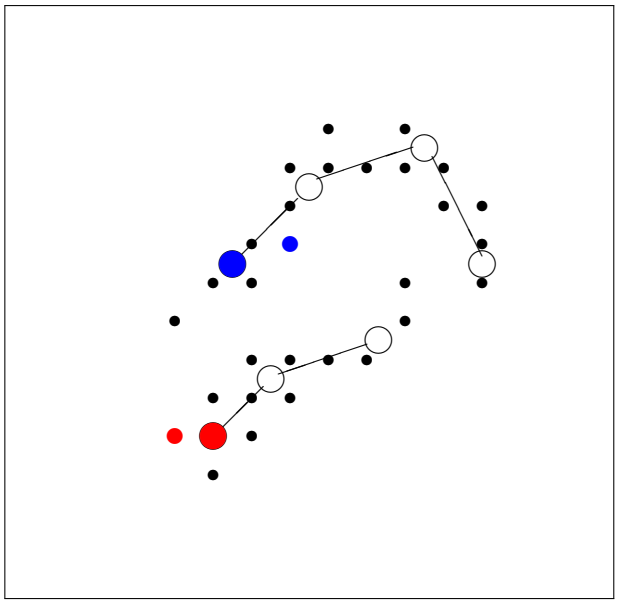


Figure 4.14: GNG, mapping the labeled data-points' labels to the vertices ($|V| = 7$, $|E| = 6$).

Chapter 5

FIR Filter Regularization approach

In this chapter we introduce the second contribution of this dissertation, which is a new SSL regularizer inspired in an application of a signal processing concept. Previously it has already been used signal processing concepts into graph applications, mainly in computer graphics [46, 47, 48]. Also there has been recent studies to formalize a signal processing approach over graphs [39, 41]. The proposed regularizer scheme is based in the application of Finite Impulse Response Filters (FIR Filters) over the built graph. The aim of this filter is to propagate the labels over the graph each time it is applied. In this chapter, firstly we introduce the concept of FIR filters, then we describe how a FIR filter works in the transductive learning with graphs scenario and finally we explain how we find the FIR filter coefficients for each graph instance.

5.1 FIR Filters

In signal processing, a filter removes and/or changes the amplitudes of frequency components of a signal [33]. There are more than one way to categorize filter types, one of those is to classify them by the duration of the impulse response of the filter: the Infinite Impulse Response (IIR) filters and the Finite Impulse Response (FIR) filters, which exist in both continuous and discrete spaces. A discrete time FIR filter is a weighted sum of the signal dislocated a finite quantity of times, Equation 5.1, where $y[n]$ is the output signal, $x[n]$ is the input signal, K is the filter order, or size, and a_i are the filter coefficients.

$$\begin{aligned} y[n] &= a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + ... + a_K x[n-K] \\ y[n] &= \sum_{i=0}^{K} a_i x[n-i] \end{aligned} \tag{5.1}$$

5.2 Transductive Learning using FIR filters

The input signal x^s of the FIR filter is a vector of size $|V| = l + u$ which has the corresponding labels of each vertex, 1 for the positive class, -1 for the negative class and 0 for the vertices with no known label ($x_i^s = 0 \; \forall \, x_i \in X_u$). The output signal y^s is also a vector of size $|V| = l+u$ with all the predicted labels, where $y_i^s = x_i^s = y_i (y_i \in Y) \; \forall \, x_i \in X_l$. The FIR filter works together with the graph Laplacian Δ and a set of coefficients a, as it is shown in Equation 5.2, where I is the identity matrix.

$$\begin{aligned} y^s &= a_0 I x^s + a_1 \Delta x^s + a_2 \Delta^2 x^s + ... + a_K \Delta^K x^s \\ y^s &= a(\Delta) x^s \\ y^s &= \sum_{i=0}^{K} a_i \Delta^i x^s ; \;\; \Delta^0 = I \end{aligned} \tag{5.2}$$

We can apply the FIR filter in a very efficient form using a recursive procedure that in each step sums a vector to a (sparse) matrix times a vector (see Equation 5.3).

$$\begin{aligned} y^s_{[0]} &= a_K x^s \\ y^s_{[1]} &= a_{K-1} x^s + \Delta y^s_{[0]} \;\; (y^s_{[1]} = a_{K-1} x^s + a_K \Delta x^s) \\ y^s_{[2]} &= a_{K-2} x^s + \Delta y^s_{[1]} \;\; (y^s_{[2]} = a_{K-2} x^s + a_{K-1} \Delta x^s + a_K \Delta^2 x^s) \\ &\vdots \\ y^s_{[K]} &= a_0 x^s + \Delta y^s_{[K-1]} \end{aligned} \tag{5.3}$$

The recursive procedure for a fast application of the FIR filter is described in Algorithm 8 and its iterative version is described in Algorithm 9.

Algorithm 8 Recursive Fast FIR

Require: $a \in \mathbb{R}^K, \Delta, x^s \in \{-1, 0, 1\}^{|V|}, i \Leftarrow K$
  if i=0 **then**
    $return \; a_K x^s$
  else
    $return \; a_{K-i} x^s + \Delta * RecursiveFastFIR(a, \Delta, x^s, i-1)$
  end if

After applying this labeling FIR filter, $y^s = f$. The filter should be applied iteratively multiple times, either by a fixed number or until y^s stops changing. In the next section we

Algorithm 9 Iterative Fast FIR

Require: $a \in \mathbb{R}^K, \Delta, x^s \in \{-1, 0, 1\}^{|V|}$
  $i \Leftarrow 1$
  $y^s \Leftarrow a_K x^s$
  while $i \leq K$ **do**
    $y^s \Leftarrow a_{K-i} x^s + \Delta y^s$
    $i \Leftarrow i + 1$
  end while
  $return\ y^s$

introduce one method to estimate the coefficients a, so to respect the constraints of $y_i^s = x_i^s = y_i \; \forall \, x_i \in X_l$.

5.3 Learning the coefficients of the FIR filter

In our problem, the optimum labeling function f should minimize Equation 5.4, where $\alpha > 0$ is a smoothing parameter. This Equation follows the same structure as Equation 3.8, with an energy function that works as regularizer.

$$\underset{f \in \mathbb{R}^{|V|}}{\arg\min}\ \alpha \sum_{i \in X_l} (f_i - x_i^s)^2 + \sum_{ij \in E} w_{ij} (f_i - f_j)^2 \tag{5.4}$$

To find the coefficients a of the FIR filter (remember that $f_i = y_i^s = \sum_{h=0}^{K} a_h (\Delta^h x^s)_i$), we rewrite Equation 5.4 as Equation 5.5.

$$\underset{a \in \mathbb{R}^K}{\arg\min}\ \alpha \sum_{i \in X_l} (f_i - x_i^s)^2 + \sum_{ij \in E} w_{ij} \left(\sum_{h=0}^{K} a_h [(\Delta^h x^s)_i - (\Delta^h x^s)_j] \right)^2 \tag{5.5}$$

In Equation 5.5, $\lambda_{ij}^h = (\Delta^h x^s)_i - (\Delta^h x^s)_j$, and the loss function also would be expressed using the FIR filter notation, as in Equation 5.6.

$$\underset{a \in \mathbb{R}^K}{\arg\min}\ \alpha \sum_{i \in X_l} \left(\left(\sum_{h=0}^{K} a_h (\Delta^h x^s)_i \right) - x_i^s \right)^2 + \sum_{ij \in E} w_{ij} \left(\sum_{h=0}^{K} a_h \lambda_{ij}^h \right)^2 \tag{5.6}$$

Using matrix notation, Equation 5.6 could be rewritten as Equation 5.7, where a is a column vector of size K containing the coefficients of the FIR filter, C_i is a column vector of size K

where each element is $(\Delta^h x^s)_i$, and λ_{ij} is another column vector of size K where each element is λ_{ij}^h.

$$\arg\min_{a\in\mathbb{R}^K} \alpha \sum_{i\in X_l} (C_i^\top a - x_i^s)^2 + \sum_{ij\in E} w_{ij}(\lambda_{ij}^\top a)^2 \tag{5.7}$$

Continuing with the regularizer and expressing the loss function as a dot product, we reach Equation 5.8, where $C = [\, C_0\ C_1\ C_2\ \cdots\ C_{l-1}\,]$, a matrix of $K \times l$ size.

$$\arg\min_{a\in\mathbb{R}^K} \alpha \left[(C^\top a - x^s)\cdot(C^\top a - x^s)\right] + a^\top \left[\sum_{ij\in E} w_{ij}\lambda_{ij}\lambda_{ij}^\top\right] a \tag{5.8}$$

Using vector magnitudes in the loss function and introducing matrix $A = \sum_{ij\in E} w_{ij}\lambda_{ij}\lambda_{ij}^T$ of size $K \times K$ in the regularizer we attain Equation 5.9, which leads to the lower dimensional linear system problem in Equation 5.10 which can be solved in closed form.

$$\arg\min_{a\in\mathbb{R}^K} \alpha\|C^\top a - x^s\|^2 + a^\top A a \tag{5.9}$$

$$(A + \alpha C C^\top)\, a = \alpha C x^s \tag{5.10}$$

Algorithm 10 Transductive FIR Learning

Require: $t, K, \alpha, W, x^s \in \{-1, 0, 1\}^{|V|}$
  $\Delta \Leftarrow calculateLaplacian(W)$
  $\Phi \Leftarrow calculateKrylovMatrix(\Delta, x^s, K)$
  $A, C \Leftarrow calculateAnC(\Phi, W)$
  $Y_l \Leftarrow getKnownLabels(x^s)$
  $a \Leftarrow calculateFIRCoefficients(\alpha, A, C, Y_l)$
  $f \Leftarrow fastFIR(a, \Delta, x^s)$
  repeat
    $f \Leftarrow fastFIR(a, \Delta, f)$
  until done t times or f no longer changes
  $return\ f$

The transductive learning algorithm over the graph using FIR filters is summarized in Algorithm 10. The FIR filter can be applied more than one time. As notion, we should apply it as many times as the greater number of edges in the graph between a labeled and an unlabeled vertices, as it propagates the the labels from the labeled data-points to the unlabeled ones.

As the FIR filter is going to be applied several times, first over x^s and then over the result of that filtering, over and over again for a while, we denote $f^0 = x^s$, being f^n the n-th estimate of f. A Krylov matrix Φ for our problem is defined in Equation 5.11. By this the $(n+1)$-th estimate of f would be $f^{n+1} = \Phi^n a$.

$$\Phi^n = \left[f^n \ \Delta f^n \ \Delta^2 f^n \ \cdots \ \Delta^K f^n \right] \tag{5.11}$$

From the Krylov matrix Φ, we can calculate the matrices A and C (Equation 5.12), where Φ_i is the i-th row (as a row vector) of the Krylov matrix Φ.

$$\begin{aligned} A &= \sum_{(ij)\in E} w_{ij} \left[(\Phi_i - \Phi_j)^\top (\Phi_i - \Phi_j) \right] \\ C &= \left[\Phi_i^\top \ \cdots \right] \ ; \forall \ i \in X_l \end{aligned} \tag{5.12}$$

Below, we present a running example of the FIR filter procedure for semi-supervised learning over the graph of Figure 5.1. This graph has 10 vertices ($|V| = 10$) and 12 edges ($|E| = 12$), it has two labeled vertices, each one from a different class ($|X_l| = 2$), one class of blue vertices and the other of red vertices. It was generating by a synthetic procedure, only for demonstrative purposes.

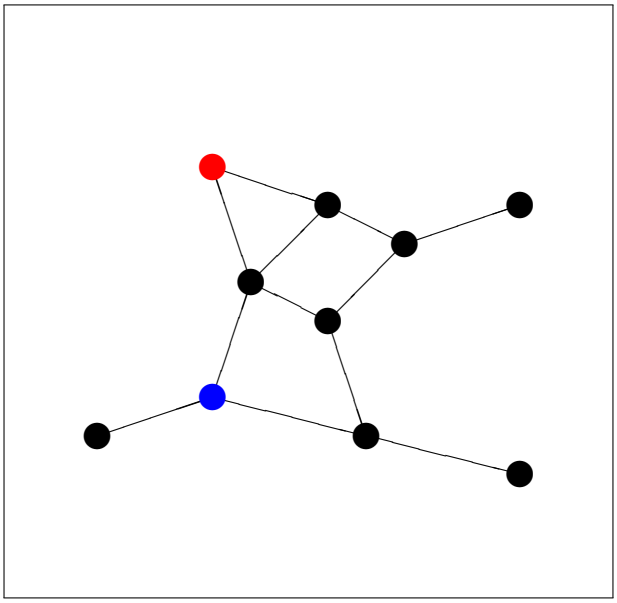


Figure 5.1: Graph representing a data set with two labeled data-points, each label from one different class ($|V| = 10$, $|X_l| = 2$, $|X_u| = 8$).

From the graph of Figure 5.1 it is constructed the signal x^s, as it is shown in Figure 5.2.

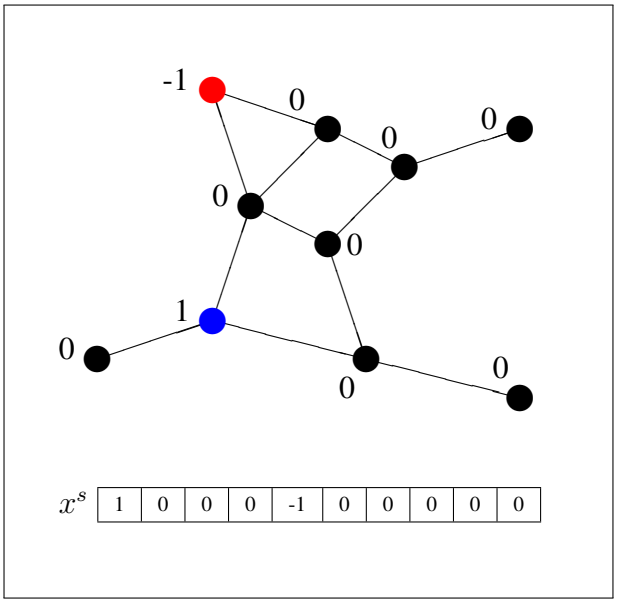


Figure 5.2: Graph and its corresponding initial signal x^s.

Then, the algorithm calculates the graph Laplacian (see Equations 3.1 and 3.2). In this case

it would a 10×10 matrix.

$$\Delta = \begin{bmatrix} \ddots & & \\ & \ddots & \\ & & \end{bmatrix}_{10 \times 10}$$

From the initial signal x^s (remember that $x^s = f^0$) and the graph Laplacian Δ, the FIR regularizer calculates the Krylov matrix Φ (see Equation 5.11). For this example it will produce a $10 \times K$ matrix, being K the filter size.

$$\Phi = \begin{bmatrix} \ddots & & \\ & \ddots & \\ & & \end{bmatrix}_{10 \times K}$$

Using the Krylov matrix Φ, it calculates the matrices A and C (see Equation 5.12). A would be a square matrix of $K \times K$, and C would be a $K \times 2$ matrix ($|X_l| = l = 2$, the number of labeled data-points).

$$C = \begin{bmatrix} \ddots & \\ & \ddots \\ & \end{bmatrix}_{K \times 2} \quad A = \begin{bmatrix} \ddots & \\ & \ddots \\ & \end{bmatrix}_{K \times K}$$

At last, we calculated the FIR filter coefficients using the matrices A, C and the input signal x^s (solving Equation 5.10). Producing a column vector of K size.

$$a = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}_{K \times 1}$$

Finally, with the column vector a, we can applied the FIR filter over the graph a finite quantity of times (using Algorithms 8 or 9). Figure 5.3, shows the signal after the filter was applied one time.

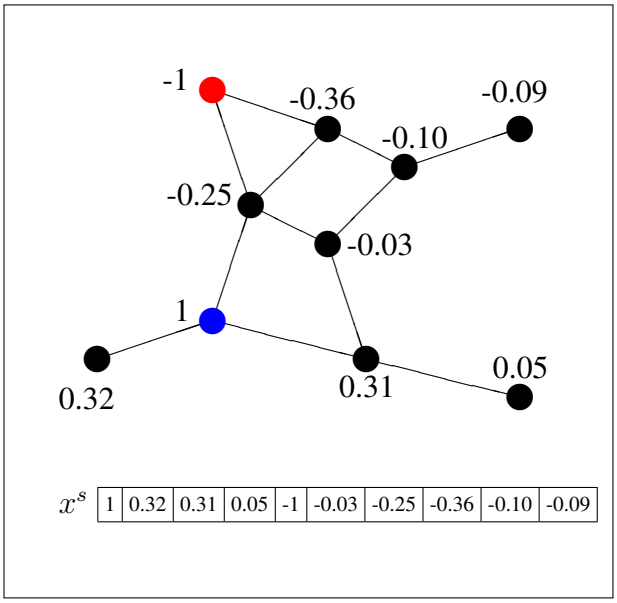


Figure 5.3: FIR regularizer, after being applied one time over a graph.

Figure 5.4 shows the signal after the filter was applied two times. The filter can be applied a predefined i number of times, or until the signal does not varies anymore.

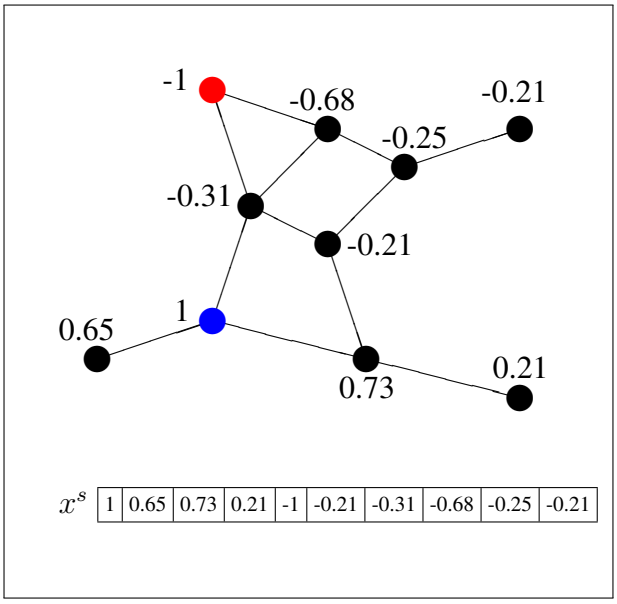


Figure 5.4: FIR regularizer, after being applied two times over a graph.

Figure 5.5 shows the signal after being processed by the FIR filter an i number of times. At

this point the vertices would have positive or negative values (not necessarily 1 or -1). Setting a threshold at 0 would make the classification.

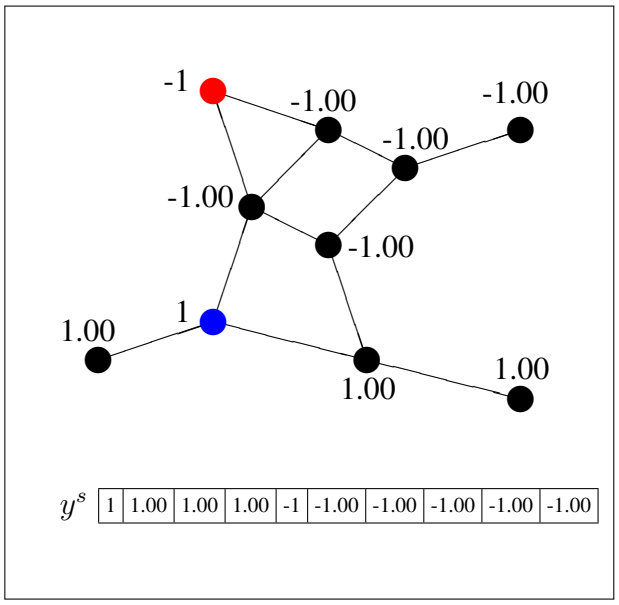


Figure 5.5: FIR regularizer, after being applied i times over a graph, and the final corresponding signal y_s.

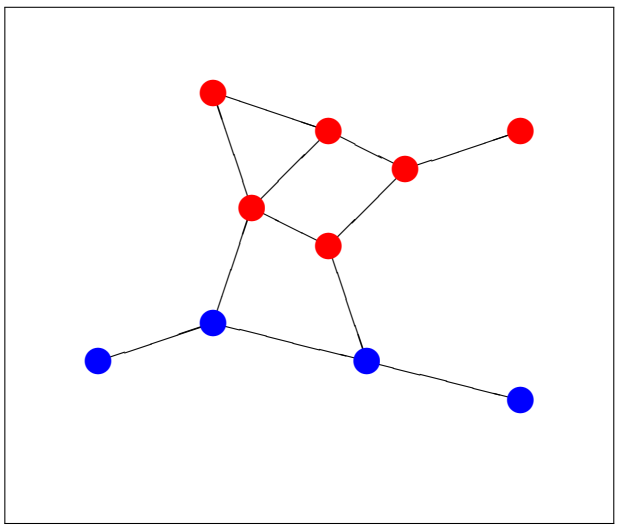


Figure 5.6: A labeled graph, processed by the FIR regularizer.

Figure 5.6 shows the result of the FIR regularizer, with all the vertices correctly labeled.

Chapter 6

Experiments and Results

In this chapter we present the experimental methodology, analysis and results, from comparing our two proposed methods to the literature ones. First we will explain the characteristic of the experimental data sets. Then we explain the experimental methodology and criteria. Then we present the results obtained from the experiments done with the GNG graph construction algorithm. After that we make a theoretical analysis of the cost of each regularization algorithm (including the FIR regularizer). Finally we present the regularization algorithms' results.

6.1 Data Sets

We use for all our experiments the data sets from the semi-supervised learning benchmark [7, 19]. This data sets encompasses artificial (g241c, g241n, Digit1) and real-world problems (COIL, BCI). The CIFAR-10 label set with the Tiny Images data set, represents a real problem where semi-supervised learning classification can be applied (when there is a lot of unlabeled data, and few labeled data), in this case in a content image classification domain [23, 50]. The Madelon data set, is an artificial and highly non-linear data set, from the UCI machine learning repository. Each data set is described in detail below. Table 6.1 summarizes the important information of them.

- **The Semi-supervised learning benchmark**. The SSL literature describes an SLL data set benchmark of 8 data sets (in this research just five of them were used) [7, 19]. Three data sets are artificial, in order to represent each of the SSL learning assumptions, the other five are derived from real world data sets. The purpose of the benchmark was to evaluate the power of the SSL algorithms themselves in a way as neutral as possible. This group of data sets were chosen because it is customary to use them in this research topic.

- **g241c** data set. This is an artificial data set that contains 1500 data-points of 241 dimensions that belong to 2 balanced classes. Each data point was built from one of two unit-variance isotropic Gaussians with some overlapping degree, the label corresponds to which Gaussian generated the data point. All dimensions are standardized, shifted and rescaled to zero-mean and unit variance. This data set holds the cluster assumption.
- **g241n** data set. This is also an artificial data set that contains 1500 data-points of 241 dimensions that belong to 2 balanced classes. Each of the data-points were generated from one of four unit-variance isotropic Gaussians. The classes are defined from which of the Gaussians they belong. Each class is represented by two of the four Gaussians, the Gaussians of the same class have their centers more separated than the center of one of the other two Gaussians of the other class.
- **Digit1** data set. This is an artificial data set that contains 1500 data-points of 241 dimensions that belong to 2 balanced classes. It was designed in order to hold the manifold assumption and not the cluster assumption. It was built from a system that generates artificial writing binary images of the digit 1 [21]. All the points were generated by randomly sampling these images (of 16×16 pixels), the label of each data point is defined by the tilt angle of the sampled images. To each image it was added some noise and they were rescaled too.
- **COIL** data set. This data set was derived from the Columbia object image library (COIL-100). Originally the images correspond to 100 different objects, each image of 128×128 pixels was down-sampled to a 16×16 image (in the red channel) [29]. Then was selected randomly 24 objects from 100, at first instance the objects were partitioned in groups of four objects, but for the binary classification scenario they were partitioned in groups of 12 objects each. Finally to each image it was added some noise and were rescaled too, leading to a data set of 1500 images of 241 dimensions in 2 balanced classes.
- **BCI** data set. This data set was originally derived from a brain computer interface data set. One single person performed movements with his left (-1 class) and right (+1 class) hands. In each trial, an electroencephalography was recorded from 39 electrodes [24]. The data set consist of 400 data-points of 117 dimensions.

- **CIFAR-10** label set with the **Tiny Images** data set. The Tiny images data set consist of 79302017 images, each being a 32×32 color image [50]. These images for our experiments were described using global image descriptors, specifically the GIST descriptor [31], which is a holistic descriptor for complete scenes of 384 dimensions. From this data set there are labels just for 60000, the CIFAR-10 label set gives 10 labels for each of

these 60000 [23] (6000 images for each label), see Figure 6.1. This data set was chosen because it was already used in the literature for SSL evaluation and also because it represent a large data set of a real problem.

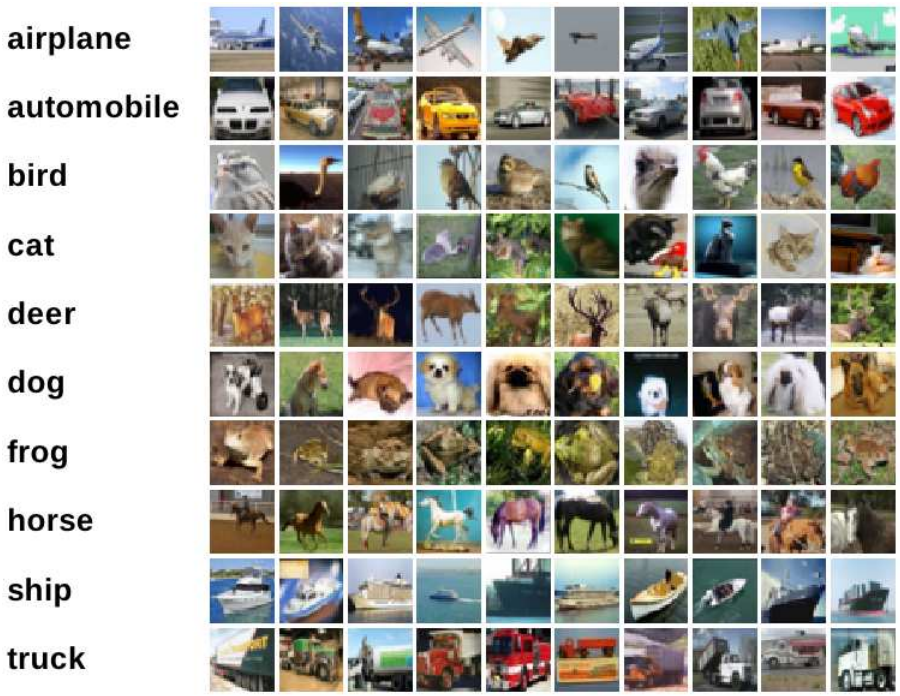


Figure 6.1: CIFAR-10 label set.

.

- **Madelon** data set. This data set was part of the NIPS feature selection challenge of 2003 [20], it consist of 4400 artificial data-points of 500 dimensions from two classes. The data set was designed from 32 clusters placed in the corners of a 5 dimensional hypercube, each cluster was randomly labeled as +1 or -1. 15 linear combinations of those 5 dimensions were added to form 20 more redundant dimensions, the rest of dimensions were added as distractor features with no predictive power. The order of the features were randomized. This data set was chosen because of its artificial nature in contrast to the Tiny images data set, and also because it represent a very difficult set to work with.

Table 6.1: Experiment's Data Sets

Data set	Labeled points	Dimensions	Classes
g241c	1500	241	2
g241n	1500	241	2
Digit1	1500	241	2
COIL	1500	241	2
BCI	400	117	2
Tiny Images	60000	384	10
Madelon	4400	500	2

6.2 Performance Evaluation Methodology and Criteria

In order to measure fairly the performance of our proposed algorithms in terms of well done classification, we are going to use an accuracy measure at different proportions of labeled data. Accuracy is a traditional performance metric in machine learning [13]. In binary classification, accuracy is the ratio of correctly classified data-points over all the data-points in the problem.

Table 6.2: Confusion matrix for a binary classification problem

	Class +	Class -
Class +	True Positives	False Negatives
Class -	False Positives	True Negatives

Given a confusion matrix for binary classification (see table 6.2) with a positive class (+) and a negative class (-), the accuracy would be given by Equation 6.1, where TP are the True Positives, TN the True Negatives, FP the False Positives and FN the False Negatives.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

From each data set we sampled five times, different labeled data, and then we propagate to all the data set the corresponding labels. We get for different label proportions 5 accuracy results, then we get the mean and the standard deviation for each label proportion and we plot the behavior of each algorithm as the labeled data proportion increases (labeled data proportion $\times$ accuracy). The idea is that the accuracy should increase as the labeled data increases, but at some point it will significantly stop increasing at the same rate. Behaviors like relative good classification with little labeled data and not as good with more labeled data, are acceptable, as these algorithms mean to work well in such conditions.

Also we test different combinations of graph construction algorithms with graph regularization algorithms, and get the result from each situation, as we explain below.

The implementation of the graph construction algorithms (from sections 3.2 and 4) and regularization algorithms (from sections 3.3, 3.4 and 5) were done using Python with the Numpy/Scipy API. The tests were done by using the RECOD laboratory's computer cluster from the Institute of Computing of the State University of Campinas, each cluster with different number of processors (from 8 to 24) and different amounts of memory (from 12 to 48 gigabytes).

6.3 Graph Construction's Performance Results

In order to validate our graph construction approach for semi-supervised learning with graphs, we compare our method with the other graph constructions algorithms: full graphs, ϵ-graphs, and k-graphs; using different graph regularization algorithms: the harmonic function regularizer (section 3.3.1), the Krylov approximated regularizer (section 3.4.1) and the eigenvectors of the smooth operator of the graph Laplacian regularizer (section 3.4.3).

For each regularization method, there can be a different graph construction method with different hyper parameters that optimize its performance. In order to discover it there has to be done performance tests or an analytical study of the data set properties. The literature points out that in theory it is better to work with fully connected graphs but there are cases when sparse graphs outperform complete connected graphs, it depends on the problem domain. Also as the GNG-graph build a graph with lesser vertices than data-points, it should at best get the same accuracy as the graph construction methods that have same number of vertices and data-points, but saving memory resources.

We choose image content classification problem for our evaluation: we want to classify a set of images by a concept (for example all the images with dogs in the foreground) from a little subset of labeled images with this concept. For this purpose we compare the graph construction-regularization combinations in different input data set size instances of the Tiny Images with CIFAR 10 data set.

The 32×32 pixel images of the Tiny Images data set, were described using the GIST descriptor of 384 dimensions. Each data point was projected on a 64 dimensional space using principal component analysis. This projection maintains 81.3 % of the original data's variance.

We have three groups of experiments, in all of them the number of known positive labeled data-points (labels from the CIFAR-10 label set) where the same as the known negative labeled data-points. The first group of experiments were made with just 1000 data-points, the second group was made with 4000 data-points and the third with 8000, each group with 5% to 50% of labeled data. We did a grid search in order to discover the hyper parameters of the GNG construction algorithm (in more detail) and the other algorithms. For the GNG algorithm the grid search was done for the following hyper parameters: p_r (the maximum number of vertices that the GNG can have), e_a (the stopping accepted error), e_f (the stopping error factor) and λ (the number of iterations needed to insert a new vertex).

As we compare the graph regularization performance with different graph construction approaches, it is necessary to know how each regularization method behaves with each graph construction method by itself. In order to accomplish this, we show the charts in Figures 6.2, 6.3 and 6.4, each one for each of the tested regularizers. In Figure 6.2 we show the performance in terms of accuracy (with 0 as decision threshold, being -1 the negative labels and 1 the positive ones) of the Harmonic regularizer with the different graph construction methods, at a progressive amount of labeled data (from 5% to 50%). The four graph methods were very tied with the Harmonic regularizer, but the best performance was obtained by the ϵ-graph ($\epsilon = 0.8$). As can be seen by this the GNG graph construction algorithm can obtain comparable results to the traditional graph construction algorithms (in some cases even wining) but saving memory, because it induces a new graph with fewer vertices, in this case was induced a graph with $0.9(l+u)$ vertices ($p_r = 0.9$) with a graph mapping index of 0.89, a GNG growing rate $\lambda = 5$, acceptance index error of 0.5 and error factor of 2.

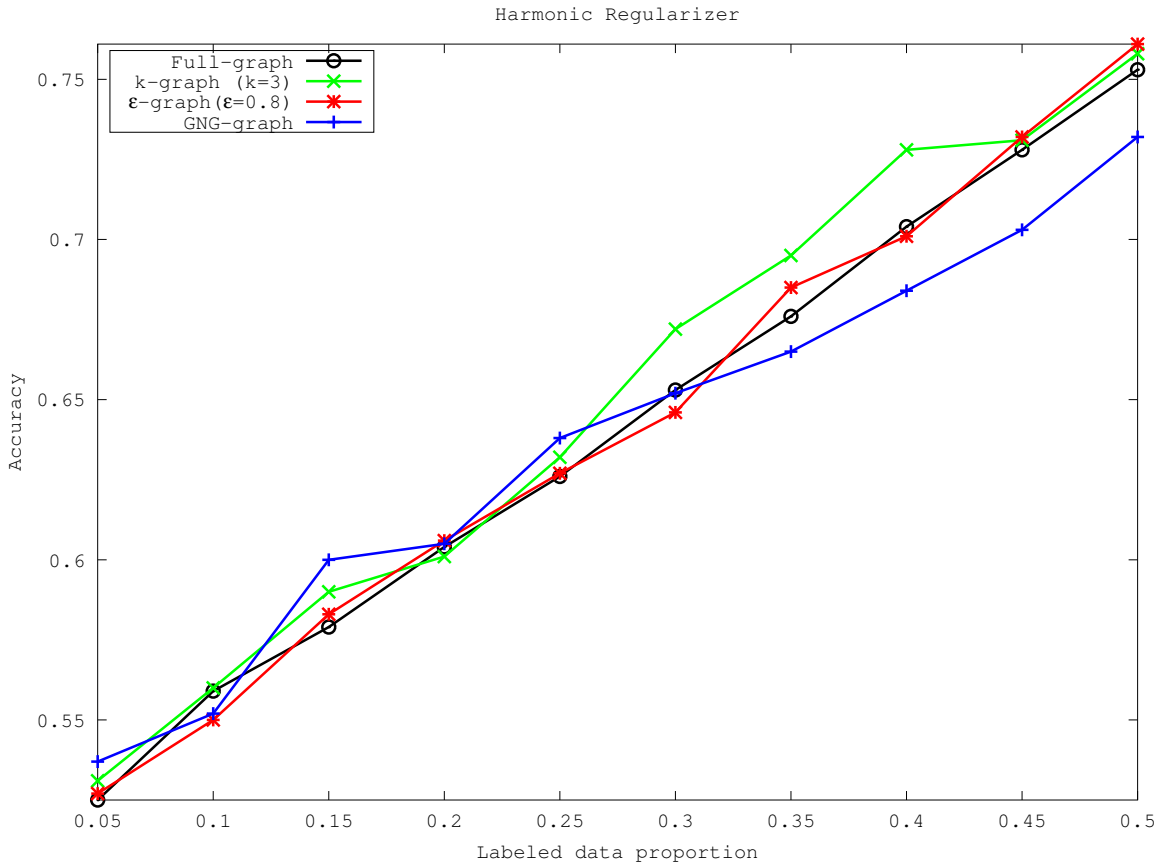


Figure 6.2: Experiment's results from the harmonic regularizer with 1000 data-points (percentage of labeled data vs accuracy).

The results from the Krylov regularizer with 1000 data-points can be seen in Figure 6.3, they are very similar to the Harmonic regularizer results. All the four graph construction methods were also tied, and the best result was obtained again by the ϵ-graph with $\epsilon = 0.8$, the experiments with the Krylov method were done with 100 iterations for the MINRES algorithm [27].

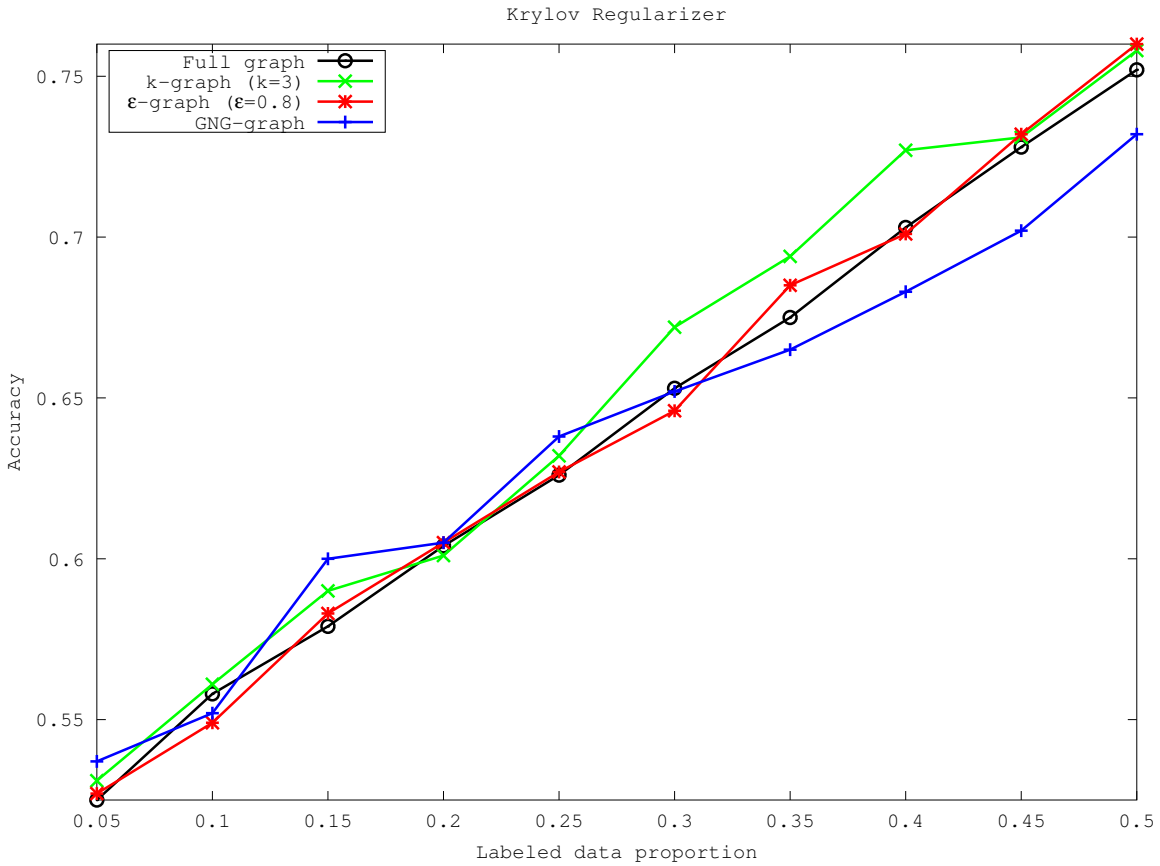


Figure 6.3: Experiment's results from the Krylov regularizer with 1000 data-points (percentage of labeled data vs accuracy).

In Figure 6.4 are shown the results from the eigenvectors of the smooth operator regularizer with $0.3(|V|)$ eigenvectors ($p = 0.3$, see Algorithm 3, this number is fixed and was chosen because it is desired to use at most a third of all eigenvectors. Remember that more eigenvectors optimize the results but do not save computational cost). In this case the results were different from those with the Harmonic regularizer and the Krylov regularizer. In these experiments the k-graph construction clearly outperforms the other three graph construction methods. But the GNG-graph reach comparable results on average, getting better results than the Full-graph and ϵ-graph in most cases. That was because, as the graph induced by the GNG graph construction algorithm is smaller ($|V| < l + u$), and the induced graph has similar spectral decomposition as the graph instances with $l + u$ vertices (pointing out the right data mapping that was accomplished by the GNG graph construction algorithm); but the new graph Laplacian of the GNG-graph has fewer representative eigenvectors with smaller eigenvalues to work with, so the proportion p is more meaningful.

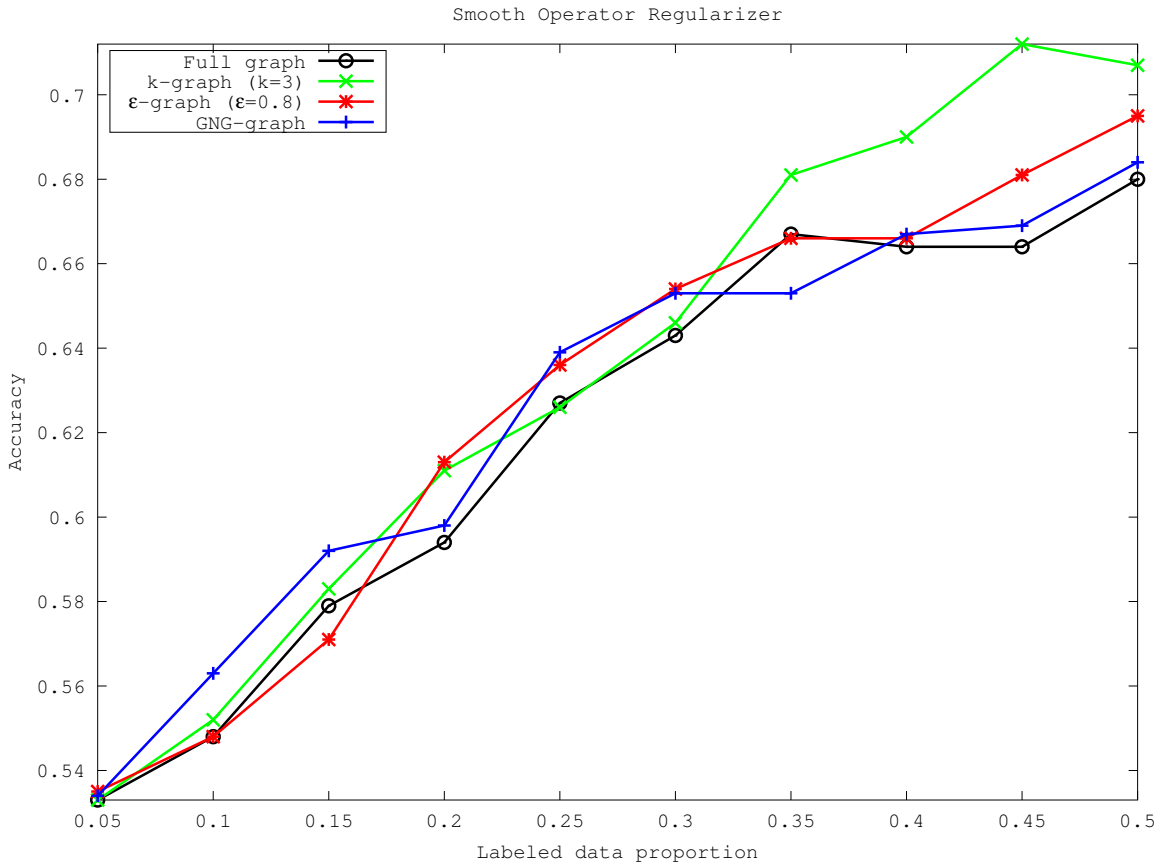


Figure 6.4: Experiment's results from the smooth operator regularizer with 1000 data-points (percentage of labeled data vs accuracy).

The detailed results of the most meaningful experiments are in tables 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 for 1000 (10% to 30% and 40% to 50%), 4000 (10% to 30% and 40% to 50%) and 8000 (10% to 30% and 40% to 50%) data-points respectively. For 8000 data-points there were induced graphs with maximum $0.8(l+u)$ vertices and an error factor of 1 in order to save space and get a faster convergence of the algorithm, and by this the accuracy was less, compared to the cases where were induced graphs with $0.9(l+u)$ vertices. The behavior is very similar in all instances, but as the input size increases the accuracy decreases in all the construction-regularization combinations.

The GNG graph construction algorithm has a complexity of $O((l+u)|V|I)$ where I is the number of executed iterations (by executed iterations we refer to each time the I_{gm} index is evaluated). Most of the implementations of the other graph construction algorithms have a complexity of $O((l+u)^2)$, so if $|V|I < (l+u)$ the GNG construction algorithm would be

faster, because it would have a better best case, but its worst case (when $|V|I > (l+u)$) could be much more expensive than $O((l+u)^2)$ if the algorithm does not converge fast or there were created too many neurons too soon and few of them are removed in the process. It can be guaranteed to always get a best case by setting as input parameters I (by setting the parameter it of Algorithm 7 as I) and $|V|$ (setting p_r, in order to $|V| = p_r|P|$), so they will be initialized with values that enforce $|V|I < (l+u)$. But if they are too small they will have a drawback in graph mapping correctness and hence classification accuracy. In order to solve these issues there have to be set appropriate values for the hyper parameters of the algorithm, and these values could vary for each data set.

From the final results of the experiments, it can be shown that the Growing Neural Gas (with stopping criteria) graph construction algorithm reduces the space and the regularization complexity with no drawbacks in terms of classification accuracy (as it induced a smaller graph), and as long as the stopping criteria works, it also reduces the required space in the graph construction step. Also in order to optimize the graph mapping index, $|V|$ has to be very close to $l+u$, but not close enough to make the GNG graph construction useless in terms of space saving.

Although the main problem with GNG construction algorithm is all the hyper parameters it has to deal with, mainly λ, the acceptance index error e_a and the error factor e_f. As we try to make the construction as fast as possible and with the better graph mapping index; λ has to be small (in this case we chose a small value as 5 for λ), by this the GNG leaves small room for adaptation, depending on $l+u$; the acceptance index error should not be too small and the error factor should not be large, by this way the GNG would stop quickly with a suitable number of vertices, which would not be greater than the proportion p_r (we used 0.9 and 0.8) of $l+u$ specified as hyper-parameter of the algorithm. As the proportion of maximum allowed vertices is lesser, depending on the topology of the data, the performance of the transductive classification could decrease, but more memory would be saved in the process; so it has to be choose a equilibrium between saved space and desired performance.

Table 6.3: Accuracy results from the 1000 data-points group (10%, 20% and 30% of labeled data).

Graph Construction	Graph Regularization	Labeled Data Proportion		
		10%	20%	30%
Full Graph	Harmonic	0.559±0.011	0.604±0.007	0.653±0.005
	Krylov	0.558±0.010	0.604±0.008	0.653±0.005
	Smooth Op.	0.548±0.009	0.594±0.013	0.643±0.034
k-Graph ($k = 2$)	Harmonic	0.548±0.006	**0.618±0.011**	0.663±0.011
	Krylov	0.548±0.006	**0.617±0.011**	0.662±0.012
	Smooth Op.	0.550±0.007	0.604±0.013	0.645±0.023
k-Graph ($k = 3$)	Harmonic	0.560±0.009	0.601±0.012	**0.672±0.006**
	Krylov	0.561±0.009	0.601±0.011	**0.672±0.006**
	Smooth Op.	0.552±0.007	**0.611±0.008**	0.646±0.023
k-Graph ($k = 5$)	Harmonic	0.562±0.018	0.604±0.014	0.661±0.009
	Krylov	0.562±0.017	0.603±0.014	0.661±0.009
	Smooth Op.	0.564±0.011	0.599±0.015	0.644±0.018
k-Graph ($k = 10$)	Harmonic	**0.564±0.009**	0.596±0.014	0.652±0.009
	Krylov	**0.563±0.009**	0.596±0.014	0.652±0.009
	Smooth Op.	**0.565±0.015**	0.601±0.010	0.637±0.013
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.550±0.000	0.600±0.000	0.650±0.000
	Krylov	0.550±0.000	0.600±0.000	0.650±0.000
	Smooth Op.	0.500±0.000	0.500±0.000	0.500±0.000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.549±0.014	0.597±0.010	0.656±0.012
	Krylov	0.549±0.014	0.597±0.010	0.656±0.012
	Smooth Op.	0.530±0.012	0.548±0.006	0.569±0.019
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.551±0.001	0.606±0.008	0.652±0.005
	Krylov	0.551±0.001	0.606±0.008	0.652±0.005
	Smooth Op.	0.543±0.009	0.596±0.010	0.636±0.019
GNG Graph ($\lambda = 5$)	Harmonic	0.552±0.012	0.605±0.015	0.652±0.014
	Krylov	0.552±0.012	0.605±0.015	0.652±0.014
	Smooth Op.	0.563±0.015	0.598±0.009	**0.653±0.014**
GNG Graph ($\lambda = 10$)	Harmonic	0.561±0.022	0.587±0.022	0.629±0.026
	Krylov	0.561±0.022	0.588±0.022	0.629±0.026
	Smooth Op.	0.551±0.015	0.587±0.011	0.604±0.022

Table 6.4: Accuracy results from the 1000 data-points group (40% and 50% of labeled data).

Graph Construction	Graph Regularization	Labeled Data Proportion	
		40%	50%
Full Graph	Harmonic	0.704±0.006	0.753±0.008
	Krylov	0.703±0.004	0.752±0.008
	Smooth Op.	0.664±0.034	0.680±0.031
k-Graph ($k = 2$)	Harmonic	0.711±0.003	0.757±0.010
	Krylov	0.711±0.003	0.756±0.011
	Smooth Op.	0.678±0.038	0.694±0.035
k-Graph ($k = 3$)	Harmonic	**0.728±0.013**	0.758±0.005
	Krylov	**0.727±0.013**	0.758±0.005
	Smooth Op.	**0.690±0.030**	**0.707±0.035**
k-Graph ($k = 5$)	Harmonic	0.722±0.006	**0.764±0.007**
	Krylov	0.723±0.006	**0.764±0.007**
	Smooth Op.	0.684±0.032	0.692±0.034
k-Graph ($k = 10$)	Harmonic	0.705±0.009	0.750±0.005
	Krylov	0.705±0.009	0.750±0.005
	Smooth Op.	0.684±0.026	0.702±0.033
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.700±0.000	0.750±0.000
	Krylov	0.700±0.000	0.750±0.000
	Smooth Op.	0.500±0.000	0.500±0.000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.701±0.010	0.740±0.007
	Krylov	0.701±0.010	0.740±0.007
	Smooth Op.	0.567±0.029	0.570±0.030
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.705±0.004	0.756±0.007
	Krylov	0.705±0.004	0.756±0.007
	Smooth Op.	0.669±0.039	0.677±0.035
GNG Graph ($\lambda = 5$)	Harmonic	0.684±0.013	0.732±0.007
	Krylov	0.683±0.013	0.732±0.007
	Smooth Op.	0.667±0.018	0.684±0.015
GNG Graph ($\lambda = 10$)	Harmonic	0.661±0.024	0.701±0.009
	Krylov	0.660±0.024	0.700±0.009
	Smooth Op.	0.635±0.029	0.648±0.011

Table 6.5: Accuracy results from the 4000 data-points group (10%, 20% and 30% of labeled data).

Graph Construction	Graph Regularization	Labeled Data Proportion		
		10%	20%	30%
Full Graph	Harmonic	0.549±0.001	0.601±0.003	0.653±0.004
	Krylov	0.549±0.001	0.601±0.003	0.652±0.004
	Smooth Op.	0.557±0.008	0.605±0.009	0.645±0.017
k-Graph ($k = 2$)	Harmonic	0.556±0.005	0.603±0.006	0.655±0.001
	Krylov	0.556±0.005	0.602±0.006	0.655±0.001
	Smooth Op.	0.557±0.005	0.601±0.008	0.641±0.021
k-Graph ($k = 3$)	Harmonic	0.554±0.007	0.601±0.005	**0.659±0.004**
	Krylov	0.554±0.007	0.601±0.005	**0.659±0.004**
	Smooth Op.	0.553±0.003	0.603±0.008	0.643±0.021
k-Graph ($k = 5$)	Harmonic	0.552±0.005	0.607±0.005	0.657±0.004
	Krylov	0.551±0.005	0.606±0.005	0.657±0.004
	Smooth Op.	0.556±0.005	0.604±0.011	0.643±0.021
k-Graph ($k = 10$)	Harmonic	**0.561±0.005**	**0.610±0.007**	0.656±0.003
	Krylov	**0.561±0.005**	**0.610±0.008**	0.656±0.003
	Smooth Op.	**0.565±0.009**	**0.607±0.008**	**0.647±0.014**
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.550±0.000	0.600±0.000	0.650±0.000
	Krylov	0.550±0.000	0.600±0.000	0.650±0.000
	Smooth Op.	0.500±0.000	0.500±0.000	0.500±0.000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.547±0.008	0.602±0.005	0.650±0.007
	Krylov	0.548±0.008	0.602±0.005	0.650±0.007
	Smooth Op.	0.516±0.004	0.534±0.002	0.546±0.003
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.550±0.003	0.599±0.001	0.648±0.002
	Krylov	0.551±0.003	0.599±0.001	0.648±0.002
	Smooth Op.	0.554±0.003	0.597±0.007	0.638±0.016
GNG Graph ($\lambda = 5$)	Harmonic	0.559±0.010	0.594±0.012	0.647±0.012
	Krylov	0.559±0.010	0.594±0.012	0.646±0.012
	Smooth Op.	0.559±0.012	0.586±0.007	0.643±0.011
GNG Graph ($\lambda = 10$)	Harmonic	0.552±0.020	0.590±0.020	0.625±0.023
	Krylov	0.552±0.020	0.590±0.020	0.625±0.023
	Smooth Op.	0.555±0.012	0.587±0.009	0.621±0.019

Table 6.6: Accuracy results from the 4000 data-points group (40% and 50% of labeled data).

Graph Construction	Graph Regularization	Labeled Data Proportion	
		40%	50%
Full Graph	Harmonic	0.700±0.004	0.748±0.003
	Krylov	0.700±0.005	0.748±0.003
	Smooth Op.	0.667±0.022	**0.687±0.028**
k-Graph ($k = 2$)	Harmonic	**0.707±0.003**	0.755±0.003
	Krylov	**0.707±0.003**	0.754±0.003
	Smooth Op.	0.662±0.024	0.680±0.026
k-Graph ($k = 3$)	Harmonic	0.705±0.003	0.759±0.004
	Krylov	0.705±0.003	0.759±0.004
	Smooth Op.	0.668±0.027	0.682±0.030
k-Graph ($k = 5$)	Harmonic	0.703±0.001	0.753±0.003
	Krylov	0.703±0.001	0.753±0.003
	Smooth Op.	0.671±0.031	0.680±0.030
k-Graph ($k = 10$)	Harmonic	0.706±0.002	**0.759±0.002**
	Krylov	0.706±0.002	**0.759±0.002**
	Smooth Op.	**0.671±0.028**	0.687±0.033
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.700±0.000	0.750±0.000
	Krylov	0.700±0.000	0.750±0.000
	Smooth Op.	0.500±0.000	0.500±0.000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.698±0.004	0.752±0.002
	Krylov	0.699±0.004	0.752±0.002
	Smooth Op.	0.561±0.007	0.578±0.013
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.699±0.003	0.746±0.002
	Krylov	0.699±0.003	0.746±0.002
	Smooth Op.	0.668±0.024	0.681±0.025
GNG Graph ($\lambda = 5$)	Harmonic	0.676±0.010	0.727±0.005
	Krylov	0.676±0.010	0.727±0.005
	Smooth Op.	0.659±0.016	0.674±0.012
GNG Graph ($\lambda = 10$)	Harmonic	0.647±0.021	0.678±0.007
	Krylov	0.648±0.021	0.679±0.007
	Smooth Op.	0.605±0.027	0.626±0.009

Table 6.7: Accuracy results from the 8000 data-points group (10%, 20% and 30% of labeled data).

Graph Construction	Graph Regularization	Labeled Data Proportion		
		10%	20%	30%
Full Graph	Harmonic	0.550±0.001	0.599±0.001	0.647±0.003
	Krylov	0.550±0.001	0.599±0.001	0.647±0.003
	Smooth Op.	**0.551±0.001**	0.599±0.003	0.642±0.003
k-Graph ($k = 2$)	Harmonic	**0.551±0.003**	0.599±0.005	**0.653±0.002**
	Krylov	**0.551±0.003**	0.599±0.005	**0.653±0.002**
	Smooth Op.	0.545±0.005	0.599±0.004	0.651±0.005
k-Graph ($k = 3$)	Harmonic	0.550±0.002	0.598±0.001	0.650±0.001
	Krylov	0.550±0.002	0.598±0.001	0.650±0.001
	Smooth Op.	0.549±0.003	0.595±0.003	**0.653±0.004**
k-Graph ($k = 5$)	Harmonic	0.549±0.005	0.600±0.001	0.650±0.001
	Krylov	0.549±0.005	0.600±0.001	0.650±0.001
	Smooth Op.	0.553±0.003	0.591±0.005	0.646±0.005
k-Graph ($k = 10$)	Harmonic	0.546±0.002	0.593±0.002	0.645±0.001
	Krylov	0.546±0.002	0.593±0.002	0.645±0.001
	Smooth Op.	0.548±0.002	**0.600±0.005**	0.642±0.005
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.550±0.000	**0.600±0.000**	0.650±0.000
	Krylov	0.550±0.000	**0.600±0.000**	0.650±0.000
	Smooth Op.	0.500±0.000	0.500±0.000	0.500±0.000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.549±0.003	0.594±0.004	0.649±0.003
	Krylov	0.548±0.001	0.595±0.005	0.649±0.003
	Smooth Op.	0.522±0.003	0.539±0.003	0.581±0.024
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.549±0.001	0.599±0.001	0.646±0.002
	Krylov	0.549±0.001	0.599±0.001	0.646±0.002
	Smooth Op.	0.548±0.002	0.597±0.001	0.652±0.003
GNG Graph ($\lambda = 5$)	Harmonic	0.538±0.010	0.590±0.015	0.637±0.010
	Krylov	0.538±0.010	0.590±0.015	0.637±0.010
	Smooth Op.	0.546±0.015	0.592±0.010	0.621±0.010
GNG Graph ($\lambda = 10$)	Harmonic	0.541±0.010	0.590±0.010	0.631±0.015
	Krylov	0.541±0.010	0.590±0.010	0.631±0.015
	Smooth Op.	0.547±0.010	0.588±0.009	0.611±0.013

Table 6.8: Accuracy results from the 8000 data-points group (40% and 50% of labeled data).

Graph Construction	Graph Regularization	Labeled Data Proportion	
		40%	50%
Full Graph	Harmonic	0.698±0.002	0.749±0.001
	Krylov	0.698±0.002	0.749±0.001
	Smooth Op.	0.677±0.004	0.688±0.004
k-Graph ($k = 2$)	Harmonic	0.698±0.003	0.749±0.002
	Krylov	0.698±0.003	0.749±0.002
	Smooth Op.	0.673±0.004	0.690±0.001
k-Graph ($k = 3$)	Harmonic	**0.702±0.001**	0.750±0.001
	Krylov	**0.702±0.001**	0.750±0.001
	Smooth Op.	0.678±0.004	0.689±0.002
k-Graph ($k = 5$)	Harmonic	0.700±0.002	**0.751±0.001**
	Krylov	0.700±0.002	**0.751±0.001**
	Smooth Op.	0.685±0.010	**0.698±0.008**
k-Graph ($k = 10$)	Harmonic	0.699±0.004	0.747±0.002
	Krylov	0.699±0.004	0.747±0.002
	Smooth Op.	0.681±0.008	0.697±0.011
ϵ-Graph ($\epsilon = 0.1$)	Harmonic	0.700±0.000	0.750±0.000
	Krylov	0.700±0.000	0.750±0.000
	Smooth Op.	0.500±0.000	0.500±0.000
ϵ-Graph ($\epsilon = 0.5$)	Harmonic	0.699±0.001	0.744±0.003
	Krylov	0.699±0.001	0.748±0.004
	Smooth Op.	0.576±0.004	0.592±0.003
ϵ-Graph ($\epsilon = 1.0$)	Harmonic	0.700±0.001	0.748±0.001
	Krylov	0.700±0.001	0.748±0.001
	Smooth Op.	**0.685±0.002**	0.696±0.003
GNG Graph ($\lambda = 5$)	Harmonic	0.670±0.013	0.699±0.007
	Krylov	0.670±0.013	0.700±0.008
	Smooth Op.	0.649±0.015	0.648±0.015
GNG Graph ($\lambda = 10$)	Harmonic	0.654±0.012	0.656±0.009
	Krylov	0.644±0.020	0.656±0.009
	Smooth Op.	0.615±0.015	0.602±0.022

6.4 Regularization Performance Results

In order to validate the proposed regularization method, we compared our algorithm to the related methods presented in sections 3.3 and 3.4, in terms of accuracy (using confidence intervals) at different labeled data proportions of the data set size. That is, we test the accuracy of our method against the others using 1%, 5%, 10%, 15%, 20%, 30% and 50% of labeled data in the training (regularization) step, sampled 5 different times for each percentage.

For running the tests, we work with the k-graph construction approach for our experiments, with $k = 5$ and $k = 10$, so as to have sparse matrices to work with. All data sets (see section 6.1), but the BCI, where projected using PCA into a lower dimensional space. The Tiny Images data set was projected maintaining 81% of the data's variance into a 64 dimensional space. The other data sets were projected maintaining 90% of the data's variance. So the g241c data-points were reduced to 193 dimensions, the g241n to 191 dimensions, Digit1 to 96, COIL to 14 and Madelon to 229 dimensions.

Table 6.9: 1% Accuracy results with 5-graph

Data Sets	Regularization Methods					
	HAR (%)	KRY (%)	SMO (%)	EIF (%)	NYS (%)	FIR (%)
g241c	50.0±3.1	50.5±0.0	50.1±2.5	50.0±0.8	**52.1±1.0**	51.1±0.8
g241n	50.0±1.4	50.5±0.1	50.5±0.7	50.3±0.7	50.2±1.4	**50.9±0.8**
Digit1	94.6±3.2	62.4±4.5	96.3±1.5	60.6±1.8	51.2±0.9	**96.8±1.2**
COIL	50.5±0.6	50.5±0.1	51.1±0.6	**60.7±3.8**	51.5±1.4	51.4±1.4
BCI	52.3±0.6	50.7±0.4	51.2±0.8	53.0±2.9	50.5±0.0	**53.6±1.8**
Tiny Images	50.4±0.3	50.7±0.6	51.5±1.2	51.1±1.6	50.4±0.6	**52.6±0.5**
Madelon	50.5±0.0	50.5±0.0	50.0±0.0	**54.4±2.9**	50.7±1.1	50.0±0.0

The hyper parameters used in the tests were: for the eigenfunction approach, 100 approximated eigenfunctions, for the smooth operator were used 90% of the eigenvectors, in the Nystrom method was 30 as sampling ratio and for the Krylov method was used 30 iterations. For our method, a grid search was done with different filter sizes and filter application times, there were done tests with 2, 3, 4, 5, 10, 15, 20 and 25 as filter size, and it was applied 5, 10, 15 and

20 times, for each test instance. The tables and figures shown the results for the best cases. The accuracy results obtained for the different data sets experiments for 1%, 5% , 15% and 50% of labeled data with a 5-graph are shown in tables 6.9, 6.10, 6.11 and 6.12 respectively.

Table 6.10: 5% Accuracy results with 5-graph

Data Sets	Regularization Methods					
	HAR (%)	KRY (%)	SMO (%)	EIF (%)	NYS (%)	FIR (%)
g241c	52.1±1.2	52.5±0.0	52.5±8.0	51.7±1.2	52.1±1.6	**53.0±1.2**
g241n	**54.0±1.3**	52.4±0.1	52.0±1.9	52.7±1.2	52.1±1.5	51.0±0.6
Digit1	96.9±1.1	96.5±1.4	97.0±1.1	74.4±1.8	52.7±1.4	**97.0±1.0**
COIL	50.9±1.4	52.5±0.0	50.3±0.7	**73.9±1.1**	52.0±1.5	50.1±0.2
BCI	52.5±3.3	52.8±0.6	51.8±2.7	50.4±2.2	53.3±2.3	**54.4±1.2**
Tiny Images	53.9±1.5	54.0±1.6	54.2±1.2	53.4±1.0	51.9±1.1	**55.1±0.7**
Madelon	52.5±0.0	52.5±0.0	50.0±0.0	**55.1±1.0**	52.3±0.8	50.1±0.1

Table 6.11: 15% Accuracy results with 5-graph

Data Sets	Regularization Methods					
	HAR (%)	KRY (%)	SMO (%)	EIF (%)	NYS (%)	FIR (%)
g241c	57.5±0.3	57.5±0.0	52.9±2.0	56.5±0.5	**60.4±1.0**	56.9±1.3
g241n	57.5±1.1	**57.5±0.1**	52.7±2.0	54.9±0.8	55.5±1.4	53.9±0.3
Digit1	98.1±0.3	98.0±0.4	**98.3±0.2**	86.6±0.6	57.2±1.5	97.3±0.8
COIL	57.1±0.7	57.7±0.0	53.6±9.4	**82.9±0.3**	56.4±1.1	53.5±0.1
BCI	56.5±1.7	57.5±0.5	**58.5±1.8**	56.8±2.3	56.0±1.2	57.3±1.4
Tiny Images	57.5±0.7	57.5±0.7	55.5±0.9	57.1±0.5	57.2±0.5	**58.1±1.3**
Madelon	**57.5±0.0**	**57.5±0.0**	53.2±0.1	54.5±0.1	57.2±0.1	54.0±0.0

Table 6.12: 50% Accuracy results with 5-graph.

Data Sets	Regularization Methods					
	HAR (%)	KRY (%)	SMO (%)	EIF (%)	NYS (%)	FIR (%)
g241c	75.0±0.4	75.0±0.0	70.5±2.1	60.6±0.4	**76.9±1.0**	70.3±1.5
g241n	**75.4±1.0**	75.0±0.1	71.2±2.2	59.7±0.9	74.7±1.3	68.5±0.3
Digit1	99.0±0.3	**99.0±0.2**	98.9±0.3	93.9±0.8	74.8±1.3	98.7±0.2
COIL	74.0±0.8	75.3±0.0	60.6±9.6	**84.3±0.3**	74.8±1.2	69.6±0.2
BCI	**77.3±1.9**	75.3±0.4	74.4±1.9	74.0±2.5	74.6±1.1	73.0±1.5
Tiny Images	**76.3±0.6**	**76.3±0.6**	74.9±0.8	62.2±0.7	75.9±0.5	72.3±0.8
Madelon	**75.0±0.0**	**75.0±0.0**	70.0±0.1	62.8±0.1	75.0±0.2	70.0±0.1

The optimum hyper parameters of the algorithm (the filter size and the number of times the filter is applied) vary from problem to problem; the better results in most cases were obtained with a filter of size 5 and applied 5 times. The experimental results show that it is not always good to apply the filter too many times. The filter should be applied as many times as the longest path between a labeled vertex to an unlabeled vertex. The learned filter coefficients are highly-dependent on the data set, a characteristic related to the similarity between two labeled vertices of different classes.

The behavior of the algorithms for each data set at different labeled data proportion can be seen in Figure 6.5, 6.6, 6.7, 6.8, 6.9 and 6.10; from there we can see that our method usually behaves better than the others when the labeled data proportion is in the low range of 1–5%, making it appropriate for situations when there is very few labeled data, such a semi-supervised ideal situation. The 10-graph's tests had very similar results to the 5-graph's test with no important differences.

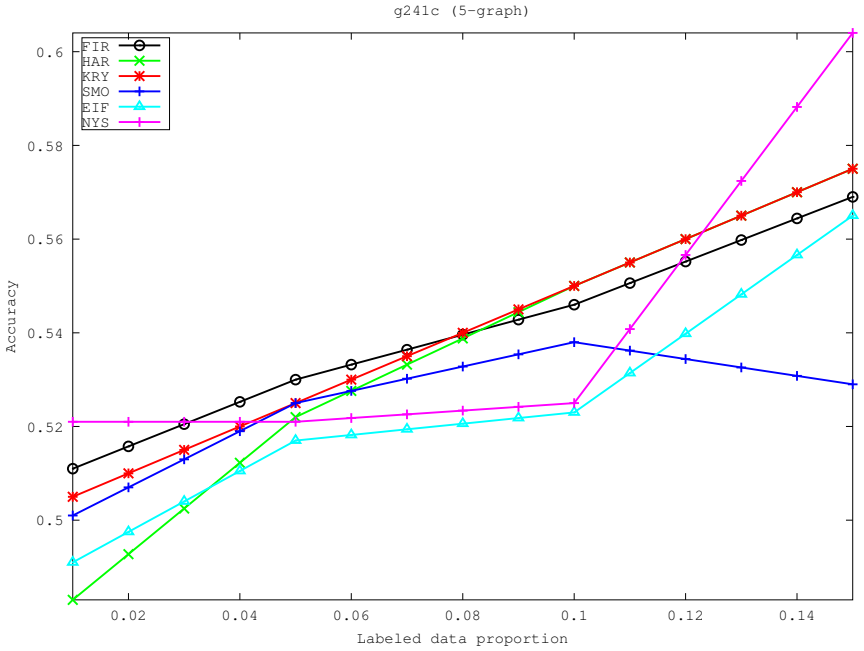


Figure 6.5: Results from the 5-graph experiments (1% to 15% of labeled data) from g241c.

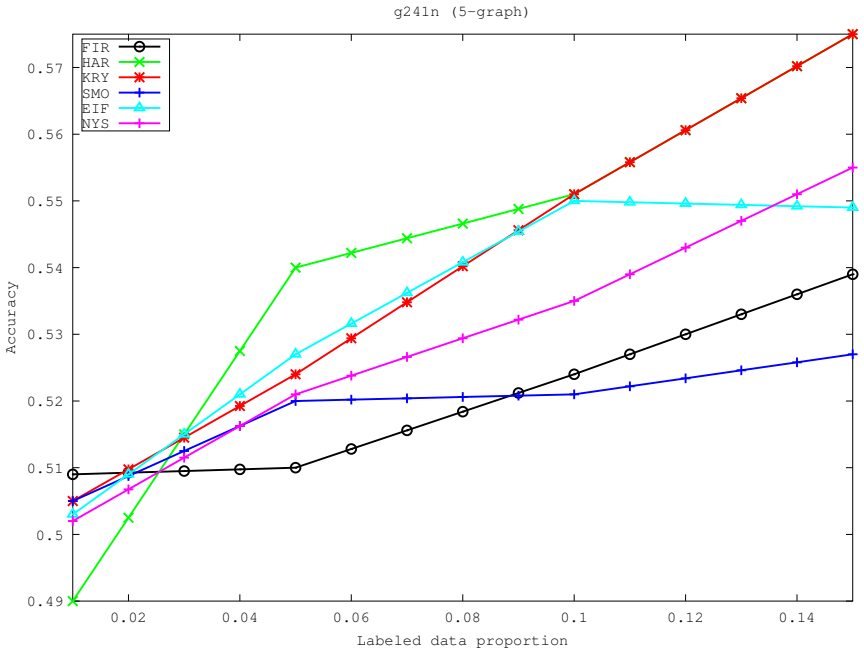


Figure 6.6: Results from the 5-graph experiments (1% to 15% of labeled data) from g241n.

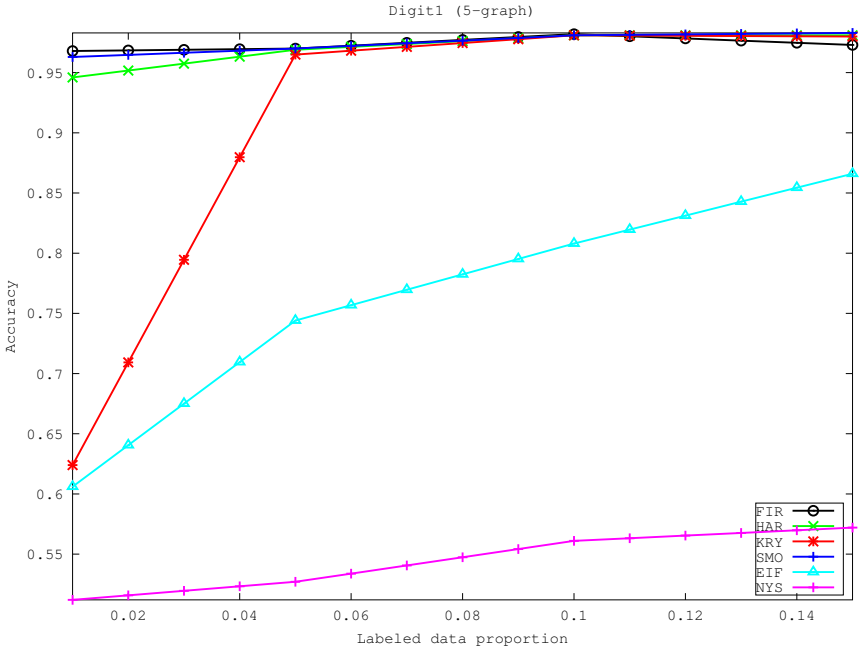


Figure 6.7: Results from the 5-graph experiments (1% to 15% of labeled data) from Digit1.

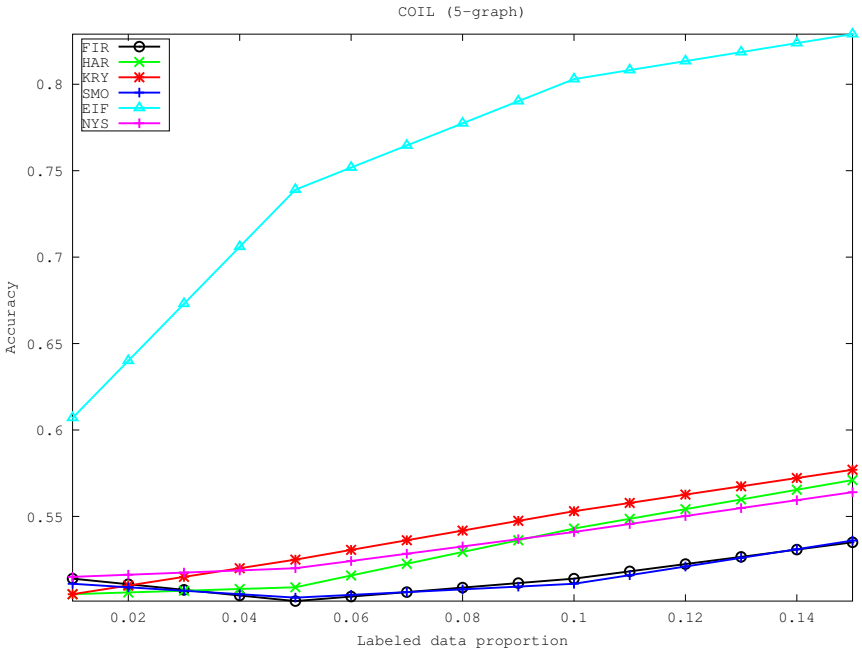


Figure 6.8: Results from the 5-graph experiments (1% to 15% of labeled data) from COIL.

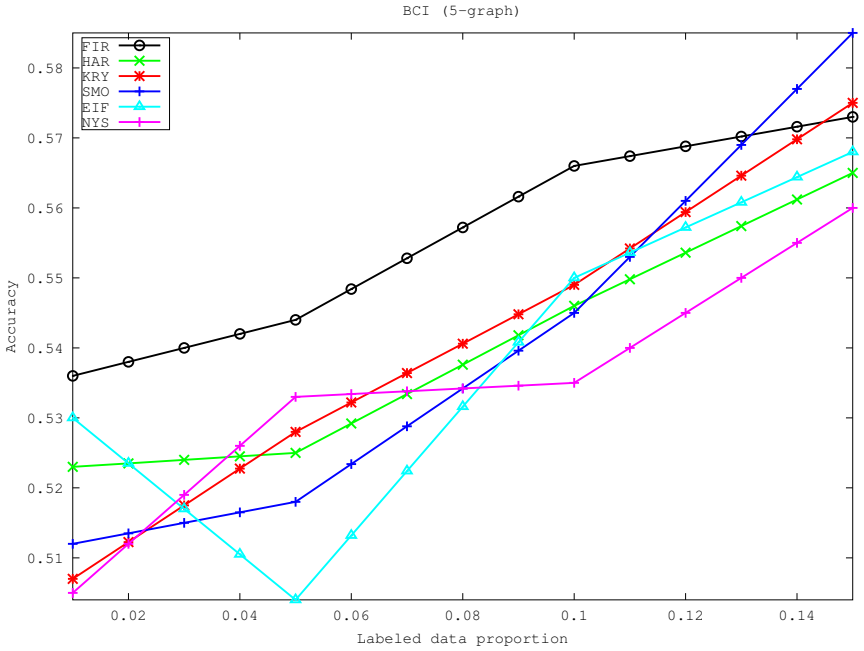


Figure 6.9: Results from the 5-graph experiments (1% to 15% of labeled data) from BCI.

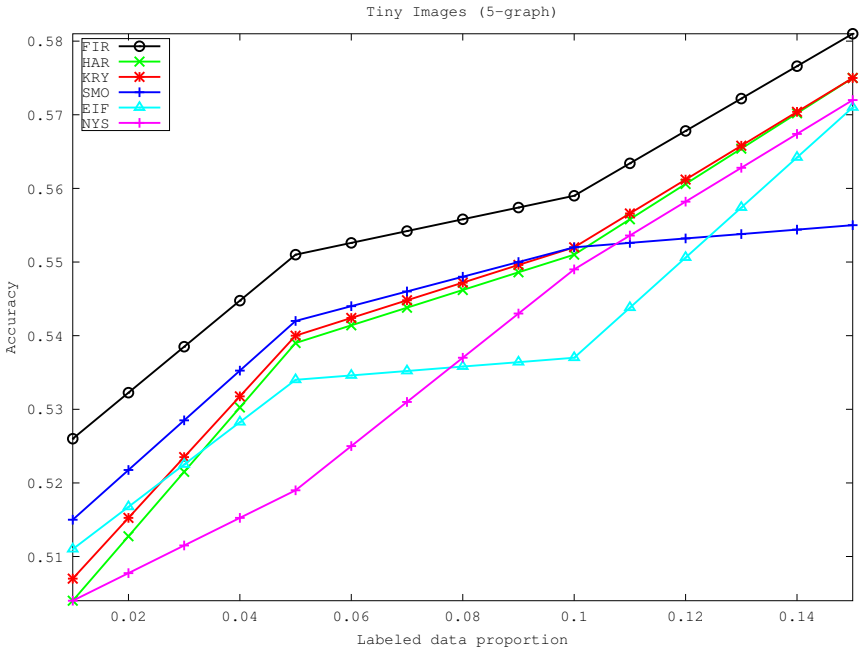


Figure 6.10: Results from the 5-graph experiments (1% to 15% of labeled data) from Tiny Images with CIFAR-10.

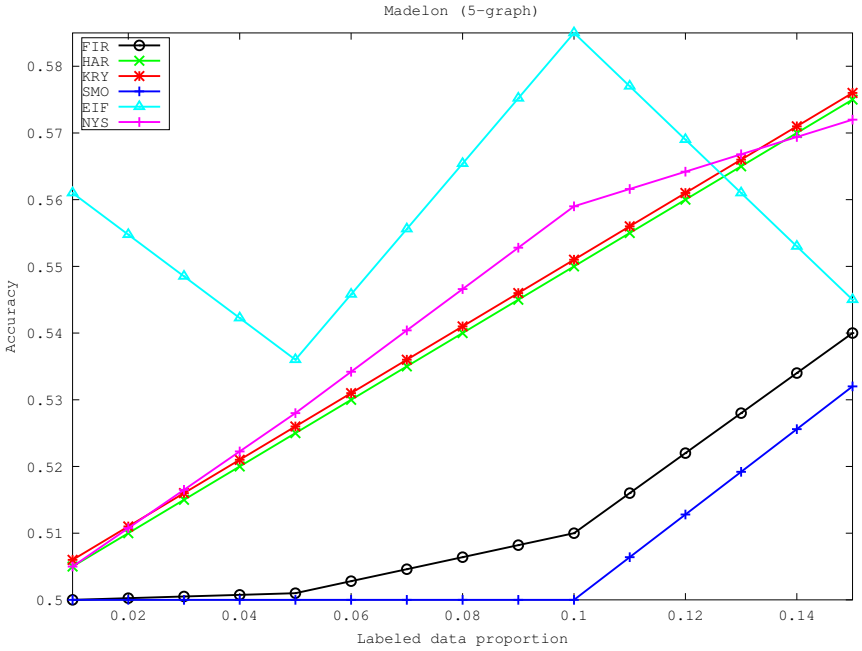


Figure 6.11: Results from the 5-graph experiments (1% to 15% of labeled data) from Madelon.

The algorithm's computational performance will depend of how many times the filter has to be applied. As the results have shown, it is not good to apply more than the largest path between a labeled vertex and an unlabeled vertex. The complexity should not increase in order to have better results. Empirically compared to the other methods, it computes at almost the same time than the Krylov and Smooth Operator regularizers, faster than the Harmonic and Nystrom method, and slower than the Eigenfunctions method (when not calculating the Laplacian explicitly).

6.5 Cost Evaluation and Discussion

Taking the execution time in each individual test as a reliable measure of cost is not viable, because of the large variation of execution time from each test instance, even when the same exact parameters and algorithms were tested, the execution time varies. All of that happens because, as the test were done in independent computer cluster, the work balance of each cluster at each time was something that can not be controlled. Table 6.13, 6.14, 6.16 and 6.17 show the regularizers' running time in seconds for 1% (g241c, g241n, Digit1 and COIL data sets), 1% (BCI, Tiny Images and Madelon data sets), 50% (g241c, g241n, Digit1 and COIL data sets) and 50% (BCI, Tiny Images and Madelon data sets) of labeled data respectively in a k-graph

with $k = 5$ (mean and standard deviation for each case). Note that the Eigenfunctions regularizer, has a significantly greater running time, because it does not need to build a graph, and the other regularizers' running times are not taking into account the graph construction running time. Table 6.15 shows the graph construction running times for a k-graph with $k = 5$, so the real time for all the tranductive learning would be the graph construction running time plus the regularization running time, except for the Eigenfunction regularizer (EIF). Tables 6.13, 6.14, 6.15, 6.16 and 6.17 show very uncorrelated numbers with large standard deviations, in many cases the standard deviations are very similar to the means. And because of this, the analysis of the running times does not give us trustful information.

Table 6.13: Regularization running times (in seconds) for 1% of labeled data in a k-graph ($k = 5$) from g241c, g241n, Digit1 and COIL data sets.

Regularizer	g241c	g241n	Digit1	COIL
FIR	170.48±37.01	304.08±88.53	88.65±60.65	23.40±7.67
HAR	141.82±90.33	99.47±46.9	47.22±45.42	55.08±35.35
KRY	143.59±97.84	119.82±41.65	145.89±116.24	67.39±24.95
NYS	874.35±187.92	868.32±152.91	548.86±474.67	1319.13±1186.29
SMO	239.07±60.31	130.72±25.3	90.44±91.22	67.41±30.26
EIF	3684.44±234.03	4771.73±607.37	414.38±305.18	6209.14±2041.5

Table 6.14: Regularization running times (in seconds) for 1% of labeled data in a k-graph ($k = 5$) from BCI, Tiny Images and Madelon data sets.

Regularizer	BCI	Tiny Images	Madelon
FIR	11.36±5.22	3135.79±1599.5	516.48±116.18
HAR	1.33±0.45	2819.27±1644.26	851.66±120.6
KRY	3.16±1.02	4995.26±2976.47	965.63±77.96
NYS	16.97±4.95	9581.66±3071.68	823.35±642.49
SMO	1.70±0.25	8941.82±5002.93	966.73±40.43
EIF	97.54±31.24	4641.78±1667.85	17299.11±1277.16

Table 6.15: Graph construction running times (in seconds) for a k-graph ($k = 5$).

g241c	g241n	Digit1	COIL	BCI	Tiny Images	Madelon
4161.56±43.8	4132.85±35.88	2612.37±176.23	6278.57±183.25	126.19±38.06	119079.13±25.02	26274.57±23.72

Table 6.16: Regularization running time (in seconds) for 50% of labeled data in a k-graph ($k = 5$) from g241c, g241n, Digit1 and COIL data sets.

Regularizer	g241c	g241n	Digit1	COIL
FIR	99.49±8.47	140.25±15.94	58.49±40.96	16.06±4.7
HAR	79.01±46.76	50.54±29.91	33.02±10.95	78.23±21.91
KRY	92.43±47.33	37.03±7.98	39.09±8.03	74.27±17.52
NYS	287.75±43.59	675.75±230.44	1918.04±743.22	1249.46±1218.09
SMO	140.14±32.08	84.58±6.05	67.63±12.61	93.32±27.43
EIF	3404.07±154.59	4040.13±387.43	1206.1±1091.97	114.29±104.17

Table 6.17: Regularization running time (in seconds) for 50% of labeled data in a k-graph ($k = 5$) from BCI, Tiny Images and Madelon data sets.

Regularizer	BCI	Tiny Images	Madelon
FIR	11.35±6.85	2717.87±340.17	180.28±21.28
HAR	21.23±1.45	2735.04±1573.17	425.69±334.24
KRY	3.95±2.14	1673.74±1351.67	470.06±396.75
NYS	14.22±12.58	6333.82±1647.48	3117.89±1547.97
SMO	3.04±1.6	2490.96±2223.06	521.97±412.94
EIF	114.15±39.41	3460.58±617.27	17068.18±2087.26

Complexities vary depending of which numeric algorithms are used to make the calculations, and are highly hyper-parameter dependent. Memory usage depends highly of the graph construction step, but not entirely. For this reason we decided to analyze both the delay of the algorithms and the memory usage by discussing which expensive steps of graph based semi-supervised learning they do (see section 3.4).

We consider that a specific regularization algorithm does an expensive step, depending of the following rules for each case:

- **Whole Graph Memory Storage**. The algorithm does this step just if they use a matrix of size $n \times n$ (being n the number of data-points, or a number very close to it, like u) that represent the graph (similarity matrix, adjacent matrix) loaded in memory.

- **Explicit Graph Construction**. If the algorithm requires by all means to process a graph built in the previous step.

- **Graph Laplacian Calculation**. If the algorithms needs to calculate the graph Laplacian from an $n \times n$ similarity matrix.

- **Matrix Multiplication**. If the algorithm multiplies two matrices of size $n \times n$, it does this step.

- **Matrix Inversion**. If the algorithm invert a matrix of size $n \times n$.

- **Matrix Spectral Decomposition**. If the algorithm needs to get the eigenvalues and eigenvectors of an $n \times n$ matrix.

Table 6.18: Expensive Steps Analysis for Regularization Methods.

Expensive Steps	HAR	KRY	NYS	SMO	EIF	FIR
Whole Graph Memory Storage	X	X	X	X	-	X
Explicit Graph Construction	X	X	X	X	-	X
Graph Laplacian Calculation	X	X	-	X	-	X
Matrix Multiplication	X	-	X	X	-	-
Matrix Inversion	X	-	-	-	-	-
Matrix Spectral Decomposition	-	-	-	X	-	-

Table 6.19: Expensive Steps Analysis for Regularization Methods using GNG-graphs.

Expensive Steps	GNG+HAR	GNG+KRY	GNG+SMO	GNG+NYS	GNG+FIR
Whole Graph Memory Storage	-	-	-	-	-
Explicit Graph Construction	X	X	X	X	X
Graph Laplacian Calculation	X	X	X	-	X
Matrix Multiplication	X	-	X	X	-
Matrix Inversion	X	-	-	-	-
Matrix Spectral Decomposition	-	-	X	-	-

Table 6.18 shows which regularization methods do each expensive step. The only studied method that does not do any of the expensive steps is the Eigenfunctions of the Graph Laplacian Regularizer (EIF) (see section 3.4.4), mainly because it does not need the graph explicitly. The two methods that do more expensivve steps are the Harmonic Regularizer (HAR) (section 3.3.1) and the Smooth Operator of the Graph Laplacian Regularizer (SMO) (section 3.4.3), both do 5 of the six expensive steps, but the SMO regularizer does a matrix spectral decomposition in order to get the eigenvectors with smallest eigenvalues in order to avoid a matrix inversion implying that the inversion is more expensive than the spectral decomposition. Our proposed method (FIR) does three expensive steps, as do the Krylov Regularizer (KRY) (section 3.4.1) and the Nystrom + Isomap Regularizer (NYS) (section 3.4.2), but the NYS regularizer has to calculate the distance within the graph of each vertex to all the other ones, making it certainly more expensive than the KRY and FIR regularizers. The KRY regularizer as the FIR regularizer are very tied in terms of costs, the KRY regularizer has to do a matrix $\times$ vector multiplication and a least square minimization in each iteration, while the FIR regularizer needs more than one matrix $\times$ vector multiplication each time the filter is applied, depending of the filter order; also it has to make a matrix inversion of a small matrix one time. Using the GNG graph construction approach, there would be no need to have the whole graph in memory (see Table 6.19), making each regularization method lighter, but affecting their performance (section 6.3).

Chapter 7

Conclusions

In this work, we studied graph based semi-supervised learning for classification, focusing our attention in fast computation algorithms. We proposed two algorithms in order to save memory, processing time and computational cost; all of that without significantly affecting the performance.

The first proposed algorithm is a graph construction algorithm, we call the graph built by it as GNG-graph, because we used a modified instance of an unsupervised neural network called Growing Neural Gas (GNG). This algorithm has as main contribution the saving in memory storage space, because it represents a data set of say n data-points with a graph of m vertices, being $m < n$, that is, the regularization step calculations can be done with significantly smaller matrices (for example 50% to 90% of the number of data-points), reducing the processing time, complexity and memory usage in later regularization stages. Moreover from the nature of this algorithm the number of edges would be even lesser than other sparse graphs, leading to more sparse matrices. The two big trade offs of this method are:

- The number of hyper parameters, most of the parameters do not have a great impact in the performance (ex: all the constants used in the vertices' error calculation and in the neuron movements calculation) or are very intuitive (ex: maximum number of neurons), but some of then can meaningfully affect the performance (ex: the error rate, the error factor and the growing rate), and are not very intuitive to set, and varies from set to set. The general idea is to have smaller graphs, so by this the error rate should not be to small and the error factor should be not to big; the big problem is given by the growing rate, this should be set as a not too small fraction of the whole data set, in order to get a better representation of the data, the growing factor should lead to a not too fast growing of the neural network.

- The processing time when the graph reaches a large size, when m is too close to n. In

early iterations the GNG graph construction algorithm works very fast, because it has few neurons to work with; but as this number increases its behavior is even slower than the k-graph and ϵ-graph, because of the KNN search and the neurons data update in each iteration. In order to deal with this problem, the maximum number of neurons and the growing rate should be set in order to not build a large graph too soon.

Generally the GNG-graph build fair representative graph with fewer neurons. The performance is not as good as with other traditional graphs, with the majority of tested regularizers, but it improves the performance of the Smooth Operator regularizer. It saves considerable memory, but as the graph grows the execution time increases; smaller graphs do not represent properly the data sets, but they save more space and their construction execution time is considerable less.

The second proposed algorithm is a regularization algorithm based in the repeated application of a FIR filter over the graph. This is done by using the graph Laplacian and a set of coefficients. The coefficients are determined by solving a minimization problem based in a regularization scheme for semi-supervised learning. This method has as main contribution the novel approach based in graph signal processing and its fast nature. It avoids matrix multiplication, inversion and decomposition for all matrices of size $n \times n$ (ex: Similarity matrix and Graph Laplacian). But it also presents some trade offs:

- The number of times that the filter has to be applied is crucial for the performance of the algorithm. This number should be close to the maximum path between an unlabeled vertex to a labeled one, calculating this path would be very expensive, so we can only make some guests in order to set it. On the other hand, as more times it is applied it will affect the execution time, but also if it is applied more times than it should be it will affect the performance.

- The method works really well when there is really small quantities of labeled data, in most cases it performs better than the other methods no matter how the data properties are. But as the labeled data proportion rises, its performance falls below some of the other methods. This property of the algorithm is very important because it goes along with the semi-supervised learning condition for classification.

In general the algorithm performance as well as the others (only beating them with very few labeled data). The execution time is similar to the others, with the exception of the Eigenfunction approach, which does not use a graph construction algorithm. Its most expensive part is the coefficient discovery stage, provided that the FIR filter is not applied too many times.

Bibliography

[1] M. F. Balcan, A. Blum, P. Pakyan, J. Lafferty, B. Pantano, M. R. Rwebangira, and X. Zhu. Person identification in webcam images: An application of semi-supervised learning. In *Proceedings of the 22ndInternational Conference on Machine Learning (ICML)*, volume 2, page 6, 2005.

[2] M. A. Belabbas and P. J. Wolfe. Spectral methods in machine learning and new strategies for very large datasets. *Proceedings of the National Academy of Sciences*, 106(2):369–374, 2009.

[3] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In *Conference on Advances in Neural Information Processing Systems*, pages 368–374, Cambridge, MA, USA, 1999. MIT Press.

[4] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 19–26, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[5] A. Blum, J. Lafferty, M. R. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 97–104, 2004.

[6] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT)*, pages 92–100, New York, NY, USA, 1998. ACM.

[7] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[8] O. Chapelle, V. Sindhwani, and S. Sathiya Keerthi. Optimization techniques for semi-supervised support vector machines. *The Journal of Machine Learning Research (JMLR)*, 9:203–233, 2008.

[9] D. Chavez, G. Laures, K. Loayza, and R. Patino. A stopping criteria for the growing neural gas based on a validity separation index for clusters. In *Proceedings of the 11th International Conference on Hybrid Intelligent Systems (HIS)*, pages 578–583, 2011.

[10] F. Gagliardi Cozman, I. Cohen, M. C. Cirelo, and Escola Politécnica. Semi-supervised learning of mixture models. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 99–106, 2003.

[11] O. Delalleau, Y. Bengio, and N. Le Roux. Efficient non-parametric function induction in semi-supervised learning. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTAT)*, pages 96–103, 2005.

[12] D. A. Chavez Escalante, G. Taubin, L. G. Nonato, and S. Klein Goldenstein. Using unsupervised learning for graph construction in semi-supervised learning with graphs. In *Proceedings of the 26th Conference on Graphics, Patterns and Images (SIBGRAPI)*, Arequipa, Peru, 2013.

[13] K. Faceli, A. C. Lorena, J. Gama, and A. C. P. L. F. de Carvalho. *Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina*. LTC, 2011.

[14] R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised learning in gigantic image collections. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[15] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(2):214–225, 2004.

[16] B. Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems*, pages 625–632. MIT Press, 1995.

[17] J. Garcke and M. Griebel. Semi-supervised learning with sparse grids. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

[18] J. Graham and J. A. Starzyk. A hybrid self-organizing neural gas based network. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 3806–3813, 2008.

[19] Y. Guo, X. Niu, and H. Zhang. An extensive empirical study on semi-supervised learning. In *Proceedings of the 10th International Conference on Data Mining (ICDM)*, pages 186–195, 2010.

[20] I. Guyon, J. Li, T. Mader, P. A. Pletscher, G. Schneider, and M. Uhr. Feature selection with the clop package. Technical report, 2006.

[21] M. Hein and J. Y. Audibert. Intrinsic dimensionality estimation of submanifolds in rd. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 289–296, New York, NY, USA, 2005. ACM.

[22] S. Kiritchenko and S. Matwin. Email classification with co-training. In *Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, pages 301–312, Riverton, NJ, USA, 2011. IBM Corp.

[23] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009.

[24] T.N. Lal, M. Schroder, T. Hinterberger, J. Weston, M. Bogdan, N. Birbaumer, and B. Schölkopf. Support vector channel selection in bci. *IEEE Transactions on Biomedical Engineering*, 51(6):1003–1010, June 2004.

[25] C. Lanczos. Solution of systems of linear equations by minimized iterations. *Journal of Research of the National Bureau of Standards*, 49:33–53, 1952.

[26] K. Li, J. Zhang, H. Xu, S. Luo, and H. Li. A semi-supervised extreme learning machine method based on co-training. In *Journal of Computational Information Systems (JCIS)*, volume 9, pages 207–214, 2013.

[27] M. Mahdaviani, N. de Freitas, B. Fraser, and F. Hamze. Fast computational methods for visually guided robots. In *International Conference on Robotics and Automation (ICRA)*, pages 138–143, 2005.

[28] T. M. Mitchell. The role of unlabeled data in supervised learning. In *Proceedings of the 6th International Colloquium on Cognitive Science*, 1999.

[29] S. Nene, S. Nayar, and H. Murase. Columbia object image library (coil-100). Technical report, Columbia University, 1996.

[30] E.J. Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

[31] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 2001.

[32] ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press, December 2013.

[33] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab. *Signals & Systems*. Prentice-Hall, Inc., 1996.

[34] C. Paige and C. Van Loan. A schur decomposition for hamiltonian matrices. *Linear Algebra and its Applications*, 41:11–32, 1981.

[35] C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *Journal on Numerical Analysis*, 12(4):617–629, 1975.

[36] J. Ratsaby and S. S. Venkatesh. Learning from a mixture of labeled and unlabeled examples with parametric side information. In *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT)*, pages 412–417, New York, NY, USA, 1995. ACM.

[37] C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *7th IEEE Workshops on Application of Computer Vision (WACV/MOTIONS)*, volume 1, pages 29–36, 2005.

[38] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37(155):105–126, 1981.

[39] A. Sandryhaila and J. M. F. Moura. Discrete signal processing on graphs. *Computing Research Repository (CoRR)*, 2012.

[40] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[41] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[42] V. Simoncini and D. B. Szyld. Theory of inexact krylov subspace methods and applications to scientific computing. *Journal on Scientific Computing*, 25:454–477, 2006.

[43] I. J. Sledge and J. M. Keller. Growing neural gas for temporal clustering. In *Proceedings of the 19th International Conference on Pattern Recognition (ICPR)*, pages 1–4, 2008.

[44] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. MIT Press, 1998.

[45] A. Talwalkar, S. Kumar, and H. Rowley. Large-scale manifold learning. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[46] G. Taubin. Curve and surface smoothing without shrinkage. In *Proceedings of the 5th International Conference on Computer Vision (ICCV)*, pages 852–857, 1995.

[47] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 351–358, New York, NY, USA, 1995. ACM.

[48] G. Taubin. Geometric signal processing on polygonal meshes. In *EUROGRAPHICS*, 2000.

[49] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[50] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(11):1958–1970, 2008.

[51] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, Inc, 1998.

[52] F. Wang and C. Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2008.

[53] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics (ACL)*, pages 189–196, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics.

[54] D. Zhou, O. Bousquet, T. Navin Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*, pages 321–328. MIT Press, 2004.

[55] X. Zhu. Semi-supervised learning literature survey. Survey, 2008.

[56] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning (ICML)*, pages 912–919, 2003.

[57] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009.

[58] Xiaojin Zhu. *Semi-Supervised Learning with Graphs*. PhD thesis, Carnegie Mellon University, 2005.