FINAL PROJECT FOR THE COURSE ADVANCED FOUNDATIONS OF MACHINE LEARNING: ONLINE LEARNING OVER GRAPHS*

PAUL SOPHER LINTILHAC† AND THOMAS NANFENG LǏ‡

MAY 9TH, 2017

1 Introduction to Graph

1.1 Concepts and Definitions

Graphs are mathematical structures that appear in many different applications of machine learning, and especially in the context of semi-supervised learning. In semi-supervised learning, we hope to exploit not only the information in the labelled examples, but also the information in the unlabelled examples by incorporating some notion of similarity between labelled and unlabelled examples. Even if our examples do not lie in a metric space, we can naturally define a notion of similarity between the examples by considering them as the vertices of a graph. One application of semi-supervised learning over graphs is in marketing and ad-tech, where we may model the flow of web traffic as a Markov Random Field. In this case, certain sites are known to be of interest to a consumer, while other sites may be of interest based on their proximity to sies previously visited on the web. Another application might be to identify suspicious accounts on a transactional network such as the block chain. In this paper we detail several different graphical models, including basic diffusion models, label propagation, regression, semi-supervised learning, and finally online learning. First, we introduce some definitions.

Let $G(V, E)$ be a graph with $V(G) = \{v_1, v_2, \cdots, v_n\}$ and $E(G) = \{e_1, e_2, \cdots, e_m\}$. The ***adjacency matrix*** of $G(V, E)$, denoted by $\boldsymbol{A}(G)$, is the $n \times n$ matrix defined as follows. The rows and the columns of $\boldsymbol{A}(G)$ are indexed by $V(G)$. If $i \neq j$ then the (i, j) entry of $\boldsymbol{A}(G)$ is 0 for vertices i and j non-adjacent, and the (i, j) entry is 1 for i and j adjacent. The (i, j) entry of $\boldsymbol{A}(G)$ is 0 for $i = j = 1, \cdots, n$. The ***degree matrix*** $\boldsymbol{D}(G)$ for $G(V, E)$ is a $n \times n$ diagonal matrix defined as

$$\boldsymbol{D}(G)_{i,j} \coloneqq \begin{cases} d(v_i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

Weighted graph $G(V, E, W)$ is a graph with real edge weights given by $w : E \to \mathbb{R}$. Here, the weight $w(e)$ of an edge e indicates the similarity of the incident vertices, and a missing edge corresponds to zero similarity. The ***weighted adjacency matrix*** $\boldsymbol{W}(G)$ of the graph $G(V, E, W)$ is defined by

$$\boldsymbol{W}_{ij} \coloneqq \begin{cases} w(e) & \text{if } e = (i, j) \in E \\ 0 & \text{otherwise.} \end{cases} . \tag{1.2}$$

The weight matrix $\boldsymbol{W}(G)$ can be, for instance, the k-nearest neighbour matrix $\boldsymbol{W}(G)_{ij} = 1$ if and only if vertex v_i is among the k-nearest neighbours of v_j or vice versa, and is 0 otherwise. Another typical weight matrix is given by the Gaussian kernel of width σ

$$\boldsymbol{W}(G)_{ij} = e^{-\frac{\|v_i - v_j\|^2}{2\sigma^2}}. \tag{1.3}$$

*New York University Courant Institute of Mathematical Sciences. Spring 2017. Professor Mehryar Mohri, Ph.D.
†New York University School of Engineering. psl274@nyu.edu
‡New York University School of Engineering. nl747@nyu.edu

Then the ***degree matrix*** $\boldsymbol{D}(G)$ for weighted graph $G(V, E, W)$ is defined by

$$\boldsymbol{D}(G)_{i,i} \coloneqq \sum_j \boldsymbol{W}(G)_{ij} \tag{1.4}$$

1.2 Graph Laplacian and Heat Equation

The graph Laplacian $\boldsymbol{L}(G)$ is defined in two different ways. The ***normalized graph Laplacian*** is

$$\boldsymbol{L}(G) \coloneqq \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{W}\boldsymbol{D}^{-\frac{1}{2}}, \tag{1.5}$$

and the ***unnormalized graph Laplacian*** is

$$\boldsymbol{L}(G) \coloneqq \boldsymbol{D} - \boldsymbol{W}. \tag{1.6}$$

The Laplacian matrix $\boldsymbol{L}$ is symmetric and positive semi-definite, and it has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. The number of 0 eigenvalues of the Laplacian matrix $\boldsymbol{L}$ is the number of connected components.

We can get more intuition about the basic objects in Graph learning by using the concept of conductance. As we saw in the previous section, the Laplacian is closely related to a second derivative, and it is no coincidence that the continuous version of the Laplacian appears in the heat equation. Let $\boldsymbol{f}$ be some real-valued function defined over the vertices of the graph. If we think of $\boldsymbol{f}$ as a hat distribution, then we can think of $\boldsymbol{L}\boldsymbol{f}$ roughly as the flux induced at each of the vertices by that distribution over the graph, based on the graph structure. Then the form $\boldsymbol{g}^T\boldsymbol{L}\boldsymbol{f}$ represents some weighted measurement of this flux with the weights given by $\boldsymbol{g}$. For example, if f is a binary vector representing the boundary of some open set on the graph, then $\langle \boldsymbol{f}, \boldsymbol{g} \rangle \coloneqq \boldsymbol{f}^T\boldsymbol{L}\boldsymbol{g}$ would be the flux produced by $\boldsymbol{g}$ as measured on the boundary represented by $\boldsymbol{f}$. We can even use this formalism to quickly prove a version of Stokes' Theorem specialized to the graph setting. In particular, we can see that the quadratic form $\boldsymbol{f}^T\boldsymbol{L}\boldsymbol{f}$ can be thought of as a standard metric for the "total flux" induced over the graph by some initial heat distribution $\boldsymbol{f}$. In other words, $\boldsymbol{f}^T\boldsymbol{L}\boldsymbol{f}$ is a measure of the smoothness of the function $\boldsymbol{f}$ relative to the structure of the graph defined by $\boldsymbol{L}$.

1.3 Label Propagation

Given the graph $G(V, E, W)$, and an initial state of some real-valued function $\boldsymbol{f} : V \to \mathbb{R}$ defined over the vertices of the graph, we may be interested in propagating this function over the graph at time steps $t = 0, 1, 2, \cdots, T$ that is consistent with the graph structure. In the simplest possible case, we might be provided with the initial state $\boldsymbol{f}_0$ and wish to see how these values propagate over the graph, as in a heat diffusion. While very simple, even scenario could be useful in settings such as fraud detection over a graph of accounts that are connected by their transactions. For example, we may have received knowledge that a small number of vertices in the graph are fraudulent, and we want to perform a quick analysis to see to what degree other vertices in the graph might be exposed to the risk emanating from these fraudulent accounts.

We proceed by deriving an update equation starting with two basic assumptions: First, we want our function $\boldsymbol{f}_{t+1}$ at each time step to be "close" to $\boldsymbol{f}_t$. Second, we want to impose the condition that $\boldsymbol{f}_{t+1}$ be a "smooth" function over the graph, in the sense of minimal total flux described above. Thus we seek to minimize

$$C\left(\boldsymbol{f}_{t+1} \middle| \boldsymbol{f}_t\right) = \left(\boldsymbol{f}_{t+1} - \boldsymbol{f}_t\right)^T \left(\boldsymbol{f}_{t+1} - \boldsymbol{f}_t\right) + \alpha \boldsymbol{f}_{t+1}^T \boldsymbol{L} \boldsymbol{f}_{t+1}, \tag{1.7}$$

where $\boldsymbol{L}$ is the non-normalized Laplacian, and α is a constant measuring the conductivity of the graph. Taking the derivative and setting it to zero yields

$$\boldsymbol{f}_{t+1} = \boldsymbol{f}_t - \alpha \boldsymbol{L} \boldsymbol{f}_t. \tag{1.8}$$

As an example, if we were to begin with the vector of initial values $\hat{\boldsymbol{Y}}^0 = e_1$ representing a unit point density, the above recurrence relation would show how that density propagates across the graph. We could alternatively consider the update equation to be a discretization of a system evolving in continuous-time. Note that the above equation can be viewed as a discrete difference equation

$$\boldsymbol{f}_{t+1} - \boldsymbol{f}_t = \delta \boldsymbol{f}_t = -\alpha \boldsymbol{L} \boldsymbol{f}_t. \tag{1.9}$$

Taking the continuous limit of this diffusion yields

$$\frac{d\boldsymbol{f}_t}{dt} = -\alpha \boldsymbol{L} \boldsymbol{f}_t, \qquad \boldsymbol{f}_t(0) = \boldsymbol{f}_0. \tag{1.10}$$

The solution to this ODE is given by

$$\boldsymbol{f}_t = e^{-\alpha \boldsymbol{L} t} \boldsymbol{f}_0. \tag{1.11}$$

We can now use this continuous system in our discretization in order to derive a discrete update formula

$$\boldsymbol{f}_t = e^{-\alpha \boldsymbol{L} t} \boldsymbol{f}_0. \tag{1.12}$$

This system differs from the one described by the original difference equation in several ways. First, there is a closed-form solution for the value of each node at any point in time, rather than a recurrence relation, which should make the time-complexity much more efficient. But it turns out that what is gained in time efficiency is more than lost inamuch higher space complexity, and thus is not practical in applications with very large graphs. Consider the case where our graph contains 100000 vertices, and all vertices are connected by a single path, with the degree of each node exactly equal to 2. In this case the regular Laplacian matrix is extremely sparse, as only 0.00001 of the entries are non-zero. Using data structures designed specifically for extremely sparse matrices, storing this matrix and performing basic computations would be quite easy. On the other hand, the matrix $e^{-\alpha \boldsymbol{L} t}$ has strictly positive entries for all t, and therefore would require 100000 times as much memory to store.

The underlying reason for this discrepancy is that whereas the continuous Laplacian establishes a dependency among all vertices within a given connected component of the graph, the discrete Laplacian $\boldsymbol{L}(G)$ only propagates to immediate neighbors, as any vertices v_i and v_j, which are not connected by an edge will have value zero. To gain more intuition about this discrepancy, we recall that the matrix exponential is defined by its Taylor series

$$e^{-\alpha \boldsymbol{L} t} = \sum_{\{i=0\}}^{\infty} \frac{(-\alpha \boldsymbol{L} t)^i}{i!}. \tag{1.13}$$

Replacing our initial condition with $\boldsymbol{f}_t$ and final state with $\boldsymbol{f}_{t+1}$ and taking the first order approximation to this series gives

$$e^{-\alpha \boldsymbol{L} t} \approx \boldsymbol{1} - \boldsymbol{L}. \tag{1.14}$$

Plugging this approximation into our update formula for the continuous case yields

$$\boldsymbol{f}_{t+1} = e^{-\alpha \boldsymbol{L}} \boldsymbol{f}_t \approx (\boldsymbol{1} - \boldsymbol{L}) \boldsymbol{f}_t = \boldsymbol{f}_t - \alpha \boldsymbol{L} \boldsymbol{f}_t, \tag{1.15}$$

which matches exactly our formula in the discrete case. If we were to take the second order approximation, each step in our iteration would take into account not only immediate neighbours, but also neighbours of neighbours and so on as more terms from the Taylor series are kept. This in turn would reduce the total number of iteration needed to simulate the evolution o the system, so that we have achieved some of the time-efficiency of the continuous model. However, we need to pay for this more favourable time-complexity by computing and storing the matrix $\boldsymbol{L}^2$. Even if $\boldsymbol{L}$ is sparse, $\boldsymbol{L}^2$will have exponentially more non-zero entries, and will thus be much larger, and will also take much longer to compute. Therefore, if we want to optimize label propagation in practice, we may want to take as many term of the Taylor expansion as we can without exceeded our memory.

1.4 Sustained Effects of Initial State

Now suppose that we wish to maintain the effects of our initial function over time. Then we may add a term to the objective function of the form $\boldsymbol{f}_{t+1}^T \boldsymbol{f}_0$. This inner product measures the similarity between the updated function $\boldsymbol{f}_{t+1}$ and the initial state vector $\boldsymbol{f}_0$. For this problem, we will also replace the non-normalized Laplacian L with the normalized Laplacian $\hat{\boldsymbol{L}}$, which will become useful when deriving the steady state solution below. Thus our objective function now has the form

$$C\left(\boldsymbol{f}_{t+1}\middle|\boldsymbol{f}_t\right) = \left(\boldsymbol{f}_{t+1} - \boldsymbol{f}_t\right)^T \left(\boldsymbol{f}_{t+1} - \boldsymbol{f}_t\right) + \boldsymbol{f}_{t+1}^T \left(\boldsymbol{I} - \alpha\hat{\boldsymbol{L}}\right) \boldsymbol{f}_{t+1} + \beta \boldsymbol{f}_{t+1}^T \boldsymbol{f}_0. \tag{1.16}$$

Straightforward calculus and linear algebra give the solution

$$\boldsymbol{f}_{t+1} = \alpha\hat{\boldsymbol{L}}\boldsymbol{f}_t + \beta\boldsymbol{f}_0. \tag{1.17}$$

The update equation derived hence agrees with that of chapter 11 of Bengio, Delalleau and Roux (2006). Following Bengio, Delalleau and Roux (2006), this recurrence relation can equivalently write the entire recurrence relation in terms of $\hat{\boldsymbol{Y}}^0$

$$\boldsymbol{f}_{t+1} = \left(\alpha\hat{\boldsymbol{L}}\right)^t \boldsymbol{f}_0 + \beta \sum_{i=0}^{t} \left(\alpha\hat{\boldsymbol{L}}\right)^i \boldsymbol{f}_0. \tag{1.18}$$

Please note that in order to write this equation in absolute form, reducing the number of computations, we need to pay for this by having to store powers of the matrix $\boldsymbol{L}$. It can be easily shown that the normalized Laplacian has an operator norm less than unity, $\left\|\hat{\boldsymbol{L}}\right\| < 1$. Therefore, the power series $\sum_{i=0}^{t} \left(\alpha\hat{\boldsymbol{L}}\right)^i$ converges, and is equal to $\left(\boldsymbol{I} - \alpha\hat{\boldsymbol{L}}\right)^{-1}$. In addition, this implies that $\lim_{t\to\infty} \left(\alpha\hat{\boldsymbol{L}}\right)^t = 0$. Thus

$$\boldsymbol{f}_\infty = \beta \left(\boldsymbol{I} - \alpha\hat{\boldsymbol{L}}\right)^{-1} . \boldsymbol{f}_0. \tag{1.19}$$

We will see in the sequel that the inverse or generalized inverse of the Laplacian and closely related matrices play a key role for online learning over graphs. The above is a linear system of equations which can be solved iteratively, as explained in section 2.

1.5 Regression on Graphs

Now that we have introduced the basic building blocks for graphical learning, we can begin to analyse graphical learning algorithms that are used in practice. Like in the previous sections, we begin by defining some cost function for our function f, and then we proceed to derive an update equation that minimizes that cost function. The sequel deals with algorithms that are actually used by machine learning practitioners. In this setting, we are given some set of labels some set of labels Y defined over the V, and we are asked to make predictions of those labels, $\hat{\boldsymbol{Y}}$.

Given a labelling $\boldsymbol{Y}$, we wish to find a set of predicted labels $\hat{\boldsymbol{Y}}$ that closely approximate the true labels, with some additional constraints that are familiar from the previous section. consistency with the initial labelling can be measured by

$$\sum_{i=1}^{l} (\hat{y}_i - y_i)^2 = \left\|\hat{\boldsymbol{Y}}_l - \boldsymbol{Y}_l\right\|^2. \tag{1.20}$$

By following the smoothness assumption, if two points x_1 and x_2 in a high-density region are close, then so should be the corresponding outputs y_1 and y_2, we consider a penalty term of the form

$$\frac{1}{2} \sum_{i,j=1}^{n} \boldsymbol{W}_{ij} \left(\hat{y}_i - \hat{y}_j\right)^2. \tag{1.21}$$

This means we penalize rapid changes in $\hat{\boldsymbol{Y}}$ between points that are close, which is given by the similarity

matrix $\boldsymbol{W}$. However, if there is noise in the available labels, it may be beneficial to allow the algorithm to relabel the labelled data. This could also help generalization in a noise-free setting where, for instance, a positive sample had been drawn from a region of space mainly filled with negative samples.

Based on this observation, Belkin, Matveeva and Niyogi (2004) proposed a more general cost criterion involving a trade-off between formula 1.20 and formula 1.21, which is shown in algorithm 1.1 and algorithm 1.2. The paper assumed that $G(V, E, W)$ is connected and that the vertices of the graph are numbered, and only partial information, $f(\boldsymbol{x}_i) = y_i$, $1 \leq i \leq k$, is given. The labels can potentially be noisy. They also allow data points to have multiplicities, i.e. each vertex of the graph may appear more than once with same or different y value. They precondition the data by mean subtracting first. That is we take

$$\tilde{\boldsymbol{y}} = (y_1 - \bar{y}, y_2 - \bar{y}, \cdots, y_k - \bar{y}), \tag{1.22}$$

where $\bar{y} = \frac{1}{k}\sum y_i$.

In the algorithm 1.1, $\boldsymbol{S}$ is a smoothness matrix, e.g. $S = L$ or $S = L^p$, $p \in \mathbb{N}$. The condition $\sum f_i = 0$ is needed to make the algorithm stable. The solution to the quadratic problem above is not hard to obtain by standard linear algebra considerations. We have the objective function $\frac{1}{k}\sum_i (f_i - \tilde{y}_i)^2 + \gamma \boldsymbol{f}^T \boldsymbol{S}\boldsymbol{f}$ and constraint $\sum f_i - 0$. Therefore, the Lagrangian is

$$\begin{aligned}
\mathcal{L} &= \frac{1}{k}\sum_i (f_i - \tilde{y}_i)^2 + \gamma \boldsymbol{f}^T \boldsymbol{S}\boldsymbol{f} - \mu\left(\sum f_i - 0\right) \\
&= \frac{1}{k}\boldsymbol{f}^T\boldsymbol{f} - \frac{2}{k}\boldsymbol{f}^T\tilde{\boldsymbol{y}} + \frac{1}{k}\tilde{\boldsymbol{y}}^T\tilde{\boldsymbol{y}} + \gamma \boldsymbol{f}^T \boldsymbol{S}\boldsymbol{f} - \mu \boldsymbol{f}^T \mathbf{1} \\
&= \boldsymbol{f}^T\left(\frac{1}{k}\boldsymbol{I} + \gamma \boldsymbol{S}\right)\boldsymbol{f} - \frac{2}{k}\boldsymbol{f}^T(\tilde{\boldsymbol{y}} + \mu\mathbf{1}) + \frac{1}{k}\tilde{\boldsymbol{y}}^T\tilde{\boldsymbol{y}}.
\end{aligned} \tag{1.23}$$

Then by taking $\frac{\partial \mathcal{L}}{\partial \boldsymbol{f}} = 0$, we have

$$\tilde{\boldsymbol{f}} = (\boldsymbol{I} + k\gamma \boldsymbol{S})^{-1}(\tilde{\boldsymbol{y}} + \mu\mathbf{1}). \tag{1.24}$$

Here, μ is chosen so that the resulting vector $\boldsymbol{f}$ is orthogonal to $\mathbf{1}$. Denote by $s(\boldsymbol{f})$ the functional

$$s : \boldsymbol{f} \to \sum_i f_i. \tag{1.25}$$

Since s is linear, we obtain

$$0 = s\left(\tilde{\boldsymbol{f}}\right) = s\left((\boldsymbol{I} + k\gamma \boldsymbol{S})^{-1}\tilde{\boldsymbol{y}}\right) + s\left((\boldsymbol{I} + k\gamma \boldsymbol{S})^{-1}\mathbf{1}\right). \tag{1.26}$$

Therefore we can write

$$\mu = -\frac{s\left((\boldsymbol{I} + k\gamma \boldsymbol{S})^{-1}\tilde{\boldsymbol{y}}\right)}{s\left((\boldsymbol{I} + k\gamma \boldsymbol{S})^{-1}\mathbf{1}\right)} \tag{1.27}$$

Algorithm 1.1 Tickhonov Regularization with Parameter $\gamma \in \mathbb{R}$

The objective is to minimize the square loss function plus the smoothness penalty.
$\tilde{\boldsymbol{f}} = \arg\min_{\substack{\boldsymbol{f}=(f_1,\cdots,f_n) \\ \sum f_i = 0}} \frac{1}{k}\sum_i (f_i - \tilde{y}_i)^2 + \gamma \boldsymbol{f}^T \boldsymbol{S}\boldsymbol{f}$

In the algorithm 1.2, $\boldsymbol{S}$ is a smoothness matrix, e.g. $S = L$ or $S = L^p$, $p \in \mathbb{N}$. However, here we are not allowing multiple vertices in the sample. We partition $\boldsymbol{S}$ as

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_1 & \boldsymbol{S}_2 \\ \boldsymbol{S}_2^T & \boldsymbol{S}_3 \end{bmatrix}, \tag{1.28}$$

where $\boldsymbol{S}_1$ is a $k \times k$ matrix, $\boldsymbol{S}_2$ is a $k \times (n-k)$ matrix, and $\boldsymbol{S}_3$ is a $(n-k) \times (n-k)$ matrix. Let $\tilde{\boldsymbol{f}}$ be the values of $\boldsymbol{f}$, where the function is unknown, $\tilde{\boldsymbol{f}} = (f_{k+1}, \cdots, f_n)$. By applying the similar Lagrangian

multiplier method, we have

$$\tilde{\boldsymbol{f}} = \boldsymbol{S}_3^{-1}\boldsymbol{S}_2^T\left(\left(\tilde{y}_1, \cdots, \tilde{y}_k\right)^T + \mu\mathbf{1}\right), \tag{1.29}$$

and

$$\mu = -\frac{s\left(\boldsymbol{S}_3^{-1}\boldsymbol{S}_2^T\tilde{\boldsymbol{y}}\right)}{s\left(\boldsymbol{S}_3^{-1}\boldsymbol{S}_2^T\mathbf{1}\right)} \tag{1.30}$$

Algorithm 1.2 Interpolated Regularization without Parameter

Here we assume that the values $y_1, y_2, \cdots, y_k$ have no noise.
Thus the optimization problem is to find a function of maximum smoothness satisfying $f\left(\boldsymbol{x}_i\right) = \tilde{y}_i, 1 \leq i \leq k$
$\tilde{\boldsymbol{f}} = \arg\min_{\substack{\boldsymbol{f}=\left(\tilde{y}_1, \tilde{y}_2, \cdots, \tilde{y_k}, f_{k+1}, \cdots, f_n\right) \\ \sum f_i = 0}} \boldsymbol{f}^T\boldsymbol{S}\boldsymbol{f}$

We now investigate generalization bounds for this graph semi-supervised learning regularization through algorithmic stability. Belkin, Matveeva and Niyogi (2004) defined the the empirical error $R_k\left(f\right)$, which is a measure of how well we do on the training set, and the generalization error $R\left(f\right)$, which is the expectation of how well we do on all labelled or unlabelled points, in the following way

$$\begin{aligned} R_k\left(f\right) &= \frac{1}{k}\sum_1^k \left(f\left(\boldsymbol{x}_i\right) - y_i\right)^2 \\ R\left(f\right) &= \mathbb{E}_\mu\left[\left(f\left(\boldsymbol{x}\right) - y\left(\boldsymbol{x}\right)\right)^2\right], \end{aligned} \tag{1.31}$$

where f_T maps a given set of examples T to $\mathbb{R}$, i.e. $f_T : V \to \mathbb{R}$, the expectation is taken over an underlying distribution μ. In theorem 5, they showed that for data samples of size $k \geq 4$ with multiplicity of at most t, γ-regularization using the smoothness functional S is a $\left(\frac{3M\sqrt{tk}}{\left(k\gamma\lambda_1 - t\right)^2} + \frac{4M}{k\gamma\lambda_1 - t}\right)$-stable algorithm, assuming that the denominator $k\gamma\lambda_1 - t$ is positive. Bousquet and Elisseeff (2001) showed that for a β-stable algorithm $T \to f_T$ we have

$$\mathbb{P}\left(\left|R_k\left(f_T\right) - R\left(f_T\right)\right| > \epsilon + \beta\right) \leq 2\exp\left(-\frac{k\epsilon^2}{2\left(k\beta + K + M\right)^2}\right), \qquad \forall\epsilon > 0. \tag{1.32}$$

Therefore, by following the derivation in the theorem 11.1 in Mohri, Rostamizadeh and Talwalkar (2012), we have with probability $1 - \delta$

$$\left|R_k\left(f_T\right) - R\left(f_T\right)\right| < \epsilon + \beta \tag{1.33}$$

where $\delta = 2\exp\left(-\frac{k\epsilon^2}{2(k\beta+K+M)^2}\right)$. We then solve for ϵ in this expression for δ and get

$$\epsilon = \sqrt{\frac{2\ln\frac{2}{\delta}}{k}}\left(k\beta + K + M\right), \tag{1.34}$$

then plug into inequality 1.33 and rearrange terms, then, with probability $1 - \delta$, we have

$$\left|R_k\left(f_T\right) - R\left(f_T\right)\right| \leq \beta + \sqrt{\frac{2\ln\frac{2}{\delta}}{k}}\left(k\beta + K + M\right), \tag{1.35}$$

where

$$\beta = \left(\frac{3M\sqrt{tk}}{\left(k\gamma\lambda_1 - t\right)^2} + \frac{4M}{k\gamma\lambda_1 - t}\right). \tag{1.36}$$

This is the theorem 1 of Belkin, Matveeva and Niyogi (2004).

2 Semi-Supervised Learning and Regularization over Graphs

Semi − supervised learning is halfway between supervised and unsupervised learning. In addition to unlabelled data, the algorithm is provided with some supervision information, but not necessarily for all

examples. Often, this information will be the targets associated with some of the examples. In this case, the data set $\boldsymbol{X} = \{x_1, x_2, \cdots, x_n\}$ can be divided into two parts: the points $\boldsymbol{X}_l = \{x_1, x_2, \cdots, x_l\}$, for which labels $\boldsymbol{Y}_l = (y_1, y_2, \cdots, y_l)$ are provided, and the points $\boldsymbol{X}_u = \{x_{l+1}, x_{l+2}, \cdots, x_{l+u}\}$, the labels of which are not known.

Building upon the previous ideas introduced in this section, we now describe a setting for semi-supervised learning over a weighted graph. We start with vertices $1, 2, \cdots, l$ labelled l with their known label1 or -1 and nodes $l+1, \cdots, n$ labelled with 0. Estimated labels on both labelled and unlabelled data are denoted by $\hat{\boldsymbol{Y}} = \left(\hat{\boldsymbol{Y}}_l, \hat{\boldsymbol{Y}}_u\right)$, where where $\hat{\boldsymbol{Y}}_l$ may be allowed to differ from the given labels $\boldsymbol{Y}_l = (y_1, y_2, \cdots, y_l)$. At each step a vertex i receives a contribution from its neighbours j and an additional small contribution given by its initial value. In addition, we wish to take into consideration that the correct labels $\boldsymbol{Y}_l$ are known for some subset of vertices.

The new semi-supervised learning objective - that we wish to minimize the difference between predicted labels and those that are known in the labelled set - can be reflected in terms of the cost function $||S\hat{Y}_v - SY_v||_2^2$, where $S = I_{[l]}$ is a selection matrix that is equal to the identity for the first l rows and columns, and equal to 0 everywhere else. In addition, since we no longer have access to the true labels for every vertex, we add a regularization term, so that we can deal with degenerate situations, such as connected components of the graph that have no labelled vertices. Thus we seek to minimize the objective function Zhou et al. (2004)

$$C(\hat{Y}) = ||S\hat{Y} - SY||_2^2 + \mu\hat{Y}L_w\hat{Y} + \epsilon\mu||\hat{Y}||_2^2. \tag{2.1}$$

We can use straightforward calculus techniques to minimize this expression

$$\frac{\partial C(\hat{Y})}{\partial \hat{Y}} = 2S(\hat{Y} - Y) + 2\mu L\hat{Y} + 2\epsilon\mu I\hat{Y} = 0. \tag{2.2}$$

Bengio, Delalleau and Roux (2006) proposed a solution to this system of linear equations based on the Jacobi iterative method for linear systems. The Jacobi iteration algorithm gives a solution to the linear system

$$Mx = b. \tag{2.3}$$

By expanding the system as $\sum_{j=1}^{n} m_{ij}x_j = b_j$. Suppose we wish to solve for x_j while assuming that all other values of x remain fixed. We decompose the sum as

$$\sum_{j=1}^{n} m_{ij}x_j = m_{ij}x_j + \sum_{k\neq j}^{n} m_{ij}x_j = b_i \rightarrow x_j = b_i - \frac{\sum_{k\neq j}^{n} m_{ij}x_j}{m_{ij}}. \tag{2.4}$$

We can write this in matrix form as

$$x^{(k)} = D^{-1}Rx^{k-1} + D^{-1}b, \tag{2.5}$$

where D is the diagonal component of M, and R is the remainder, (i.e. D+R=M). In our case, $M = S+\mu L+\epsilon\mu I$, and $b = SY$. The Jacobi method is particularly well suited for the graphical setting, since $M = S+\mu D+\epsilon\mu I$, and $R = \mu W$. A sufficient condition for the convergence of the Jacobi method is when the matrix M is strictly diangonally dominany, meaning that the absolute value of each diagonal entry is greater than any other entry in the same row or column. Since the diagonal entries of the Laplacian are by definition positive and at least as great as any entry in the same row or column, adding the positive diagonal matrix $S + \epsilon\mu I$ to μL makes $M = S + \mu L + \epsilon\mu I$ strictly diagonally dominant, as desired.

3 Projection Algorithm for Graph Online Learning

A different approach was proposed by Herbster, Pontil and Wainer (2005) using projection and kernel methods. The method uses minimal norm interpolation, which I will later explain can be thought of as finding the distribution over the vertices that minimizes the total flux induced over the graph, under the constraint

Algorithm 2.3 Jacobi Iterative Label Propagation Algorithm

Compute weight matrix $\boldsymbol{W}$ from formula **??** such that $\boldsymbol{W}_{ii} = 0$
Compute the diagonal degree matrix $\boldsymbol{D}$ by $\boldsymbol{D}_{ii} = \sum_j \boldsymbol{W}_{ij}$
Choose a parameter $\alpha \in (0,\, 1)$ and a small $\epsilon > 0$
$\mu = \frac{\alpha}{1-\alpha} \in (0,\, +\infty)$
Compute the diagonal matrix $\boldsymbol{A}$ by $\boldsymbol{A}_{ii} = \boldsymbol{I}_l\,(i) + \mu \boldsymbol{D}_{ii} + \mu\epsilon$
Initialize $\hat{\boldsymbol{Y}}^{(0)} = (y_1,\, y_2,\, \cdots,\, y_l,\, 0,\, 0,\, \cdots,\, 0)$
Iterate
$\quad \hat{\boldsymbol{Y}}^{(t+1)} = \boldsymbol{A}^{-1}\left(\mu \boldsymbol{W} \hat{\boldsymbol{Y}}^{(t)} + \hat{\boldsymbol{Y}}^{(0)}\right)$
until convergence to $\hat{\boldsymbol{Y}}^{(\infty)}$
Label point v_i by the sign of $\hat{y}_i^{(\infty)}$

that the labelled points are approximately correct.

3.1 Pseudo-Inverse of Graph Laplacian and Kernel of Hilbert

A key object in this approach is the pseudo-inverse of the graph Laplacian $\boldsymbol{L}^+$ can be thought of as the reproducing kernel of a particular Hilbert space H of real-valued functions over the vertices of the graph

$$\boldsymbol{f} : V \to \mathbb{R}^n \tag{3.1}$$

equipped with the inner product

$$\langle \boldsymbol{f},\, \boldsymbol{g} \rangle = \boldsymbol{f}^T \boldsymbol{L} \boldsymbol{g}. \tag{3.2}$$

Over the entire Vector space of real-valued functions on the graph, $\langle \boldsymbol{f},\, \boldsymbol{g} \rangle$ is only a semi-inner product, and therefore induces the semi-norm $\|\boldsymbol{g}\|$. If $\boldsymbol{f}^*$ is an eigenvector of $\boldsymbol{L}$ corresponding to an eigenvalue of 0, then $\langle \boldsymbol{f}^*,\, \boldsymbol{f}^* \rangle = 0$. These eigenvectors with eigenvalues of zero are precisely the eigenvectors that are piecewise constant. We can see this by noting that

$$\|\boldsymbol{g}\|^2 = \sum_{(i,\,j) \in E(G)} (\boldsymbol{g}_i - \boldsymbol{g}_j)^2 \quad . \tag{3.3}$$

This is because

$$\begin{aligned}
\|\boldsymbol{g}\|^2 &= \langle \boldsymbol{g},\, \boldsymbol{g} \rangle \\
&= \sum_{i,\,j=1}^{n} \boldsymbol{g}_i \boldsymbol{L}_{ij} \boldsymbol{g}_j \\
&= \sum_{i=j}^{n} \boldsymbol{g}_i \boldsymbol{L}_{ij} \boldsymbol{g}_j + \sum_{i \neq j}^{n} \boldsymbol{g}_i \boldsymbol{L}_{ij} \boldsymbol{g}_j \\
&= \sum_{i=1}^{n} \boldsymbol{g}_i^2 \boldsymbol{D}_{ii} + \sum_{i \neq j}^{n} \boldsymbol{g}_i \boldsymbol{A}_{ij} \boldsymbol{g}_j \\
&= \sum_{i,\,j=1}^{n} \boldsymbol{g}_i{}^2 \boldsymbol{A}_{ij} + \sum_{i \neq j}^{n} \boldsymbol{g}_i \boldsymbol{L}_{ij} \boldsymbol{g}_j \\
&= \sum_{i,\,j \in E(G)}^{n} \boldsymbol{g}_i^2 - 2\boldsymbol{g}_i \boldsymbol{g}_j \\
&= \sum_{(i,\,j) \in E(G)} (\boldsymbol{g}_i - \boldsymbol{g}_j)^2.
\end{aligned} \tag{3.4}$$

Please notice that the diagonal component of $\boldsymbol{L}$ is $\boldsymbol{D}$, the off-diagonal component of $\boldsymbol{L}$ is $-\boldsymbol{A}$, $\boldsymbol{D}_{ii} = \sum_{j=1}^{n} \boldsymbol{A}_{ij}$, and in the proof we pick up a factor of 2 from the second summation since E is the set of unordered pairs in the edge set, and therefore each edge gets counted twice.

As we can see, the sum vanishes if $\boldsymbol{g}_i = \boldsymbol{g}_j$ for all vertices connected by an edge, i.e. when g is piecewise constant on each connected component. Therefore we must restrict our functions to be in the subspace of

H spanned by the eigenvectors that have non-zero eigenvalues. Restricted to this subspace, $\langle \boldsymbol{f}, \boldsymbol{g} \rangle$ is indeed an inner product, and therefore induces a true norm $\|\boldsymbol{g}\|$. Since H is a Hilbert space, we can use the Riesz Representation theorem to conclude that the evaluation functional of $\boldsymbol{f}$ at any vertex, $\boldsymbol{f}(v_i)$, being a linear functional $(\boldsymbol{f}+\boldsymbol{g})(v_i) = (\boldsymbol{f}+\boldsymbol{g})_i(v) = \boldsymbol{f}_i(v) + \boldsymbol{g}_i(v) = \boldsymbol{f}(v_i) + \boldsymbol{g}(v_i)$, can be represented as an inner product

$$\boldsymbol{f}(v_i) = \langle \boldsymbol{K}_i, \boldsymbol{f} \rangle = \boldsymbol{K}_i \boldsymbol{L} \boldsymbol{f}. \tag{3.5}$$

The pseudo-inverse $\boldsymbol{L}^+$ satisfies this property, which can be verified by noting that

$$\boldsymbol{f} = \boldsymbol{f}_i = \boldsymbol{L}_i^+ \boldsymbol{L} \boldsymbol{f}_i. \tag{3.6}$$

Thus $\boldsymbol{L}^+$ is the reproducing kernel of H.

3.2 Online Projections over Graphs

With that in mind, we can introduce the mathematical framework for the online projection algorithm of Herbster, Pontil and Wainer (2005). Herbster seeks to minimize the number of mistakes (cumulative error) on the sequence of pattern-label pairs over a series of trials. The key tool for this algorithm is projection, which is defined by definition 3.1.

Definition 3.1. The projection of a point $\boldsymbol{w} \in \mathcal{H}$, which is a Hilbert space with inner product $\langle \cdot, \cdot \rangle$ and norm $\|\cdot\| = \sqrt{\langle \cdot, \cdot \rangle}$, onto a closed convex non-empty set $\mathcal{N} \subseteq \mathcal{H}$ is defined by

$$P(\mathcal{N}; \boldsymbol{w}) \coloneqq \arg\min_{\boldsymbol{u} \in \mathcal{N}} \|\boldsymbol{u} - \boldsymbol{w}\|. \tag{3.7}$$

From the definition, we can see that $\boldsymbol{u}$ any point in $\mathcal{N}$ that is closest to $\boldsymbol{w}$. If we assume $\boldsymbol{u}^* = P(\mathcal{N}; \boldsymbol{w}) = \arg\min_{\boldsymbol{u} \in \mathcal{N}} \|\boldsymbol{u} - \boldsymbol{w}\|$, then the solution is characterized by an orthogonality condition: $\boldsymbol{u}^*$ is the solution if and only if

$$\langle \boldsymbol{w} - \boldsymbol{u}^*, \boldsymbol{u} - \boldsymbol{u}^* \rangle = 0. \tag{3.8}$$

Therefore by the Pythagorean Theorem, we have

$$\|\boldsymbol{u} - \boldsymbol{w}\|^2 = \|\boldsymbol{u} - \boldsymbol{u}^*\|^2 + \|\boldsymbol{u}^* - \boldsymbol{w}\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}\|^2. \tag{3.9}$$

Hence we have the following theorem.

Theorem 3.1. *If $\mathcal{N}$ is a closed convex subset of $\mathcal{H}$, then, for every $\boldsymbol{u} \in \mathcal{N}$ and $\boldsymbol{w} \in \mathcal{H}$, we have that*

$$\|\boldsymbol{u} - \boldsymbol{w}\|^2 \geq \|\boldsymbol{u} - P(\mathcal{N}; \boldsymbol{w})\|^2 + \|P(\mathcal{N}; \boldsymbol{w}) - \boldsymbol{w}\|^2. \tag{3.10}$$

In particular, if $\mathcal{N}$ is an affine set the equality holds.

Based on theorem 3.1, and we let $\mathcal{D} = \{(x_i, y_i)\}_{t=1}^l \subseteq \mathcal{X} \times \{-1, +1\}$ be an example set, let $\mathcal{D}_t$denote the first t examples of $\mathcal{D}$. if we consider an online learning algorithm (algorithm 3.4) such that at each iteration, $\boldsymbol{w}_t$ is updated by the projection 3.7, then it is convergent. Because on any trial $t = 1, 2, \cdots, l$, the theorem 3.1 implies, for all $\boldsymbol{u} \in \mathcal{U}$, that $\|\boldsymbol{u} - \boldsymbol{w}_t\|^2 \geq \|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|^2 + \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2$, which is $\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 + \|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|^2$. Therefore, by summing it from $t = 1$ to $t = l$, we have $\sum_{t=1}^l \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{l+1}\|^2$, where $\mathcal{U}_t \in \mathcal{H}$, and $\boldsymbol{u} \in \cap_{t=1}^l \mathcal{U}_t$.

Algorithm 3.4 Prototypical Projection Algorithm

Input: A sequence of closed convex sets $\{\mathcal{U}_t\}_{t=1}^l \subset \mathcal{H}$
Initialization: $\boldsymbol{w}_1 \in \mathcal{H}$
for $t = 1, 2, \cdots, l$ **do**
    $\boldsymbol{w}_{t+1} = P(\mathcal{U}_t; \boldsymbol{w}_t)$
end

Lemma 3.1. *If* $(x,\, y) \in \mathcal{H} \times \{-1,\, +1\}$ *and* $\boldsymbol{w} \in \mathcal{H}$*, then*

$$P\left(\{\boldsymbol{u} :\, \langle \boldsymbol{u},\, \boldsymbol{x}\rangle = y\};\, \boldsymbol{w}\right) = \boldsymbol{w} + \frac{(y - \langle \boldsymbol{w},\, \boldsymbol{x}\rangle)}{\|\boldsymbol{x}\|^2}\boldsymbol{x} \tag{3.11}$$

$$P\left(\{\boldsymbol{u} :\, y\,\langle \boldsymbol{u},\, \boldsymbol{x}\rangle \geq 1\};\, \boldsymbol{w}\right) = \boldsymbol{w} + \frac{y \max\left(0,\, 1 - y\,\langle \boldsymbol{w},\, \boldsymbol{x}\rangle\right)}{\|\boldsymbol{x}\|^2}\boldsymbol{x} \tag{3.12}$$

Proof. We give a proof of formula 3.11, which is almost exactly the same as the proof of formula 3.12. We know that $\{\boldsymbol{u} :\, \langle \boldsymbol{u},\, \boldsymbol{x}\rangle = y\} = \left\{\boldsymbol{u} :\, \left\langle \boldsymbol{u} - \frac{\boldsymbol{x}y}{\|\boldsymbol{x}\|^2},\, \boldsymbol{x}\right\rangle = 0\right\}$. Now let $\boldsymbol{v}(\boldsymbol{u}) = \boldsymbol{u} - \frac{\boldsymbol{x}y}{\|\boldsymbol{x}\|^2}$. Then we can find this projection of $\boldsymbol{w}$ onto $\{\boldsymbol{u} :\, \langle \boldsymbol{u},\, \boldsymbol{x}\rangle = y\}$ by first projecting $\boldsymbol{v}(\boldsymbol{w})$ onto $\{\boldsymbol{u} :\, \langle \boldsymbol{u},\, \boldsymbol{x}\rangle = 0\}$, $\boldsymbol{v}^* = \mathrm{Proj}\left(\{\boldsymbol{u} :\, \langle \boldsymbol{u},\, \boldsymbol{x}\rangle = 0\};\, \boldsymbol{v}(\boldsymbol{w})\right)$ and then taking $\boldsymbol{w}^* = \boldsymbol{v}^{-1}(\boldsymbol{v}^*)$.

$$\begin{aligned}
\boldsymbol{v}^* &= \mathrm{Proj}\left(\{\boldsymbol{u} :\, \langle \boldsymbol{u},\, \boldsymbol{x}\rangle = 0\};\, v(\boldsymbol{w})\right) \\
&= \boldsymbol{v}(\boldsymbol{w}) - \mathrm{Proj}_x\left(v(\boldsymbol{w})\right) \\
&= \boldsymbol{v}(\boldsymbol{w}) - \frac{\langle v(\boldsymbol{w}),\, \boldsymbol{x}\rangle}{\langle \boldsymbol{x},\, \boldsymbol{x}\rangle}\boldsymbol{x} \\
&= \boldsymbol{v}(\boldsymbol{w}) + \frac{\left\langle \boldsymbol{w} - \frac{\boldsymbol{x}y}{\|\boldsymbol{x}\|^2},\, \boldsymbol{x}\right\rangle}{\langle \boldsymbol{x},\, \boldsymbol{x}\rangle}\boldsymbol{x} \\
&= \boldsymbol{v}(\boldsymbol{w}) + \frac{y - \langle \boldsymbol{w};\, \boldsymbol{x}\rangle}{\langle \boldsymbol{x},\, \boldsymbol{x}\rangle}\boldsymbol{x}.
\end{aligned} \tag{3.13}$$

Thus

$$\boldsymbol{w}^* = \boldsymbol{w} + \frac{(y - \langle \boldsymbol{w},\, \boldsymbol{x}\rangle)}{\|\boldsymbol{x}\|^2}\boldsymbol{x} \tag{3.14}$$

□

Based on the prototypical projection algorithm 3.4 and lemma 3.1, the main algorithm of Herbster, Pontil and Wainer (2005) is given in algorithm 3.5. In the algorithm, M is the set of trials in which the algorithm predicts incorrectly, U_t is the index set, which is determines by the ***strategy*** function on trial t. This in turn determines the ***noncyclic*** feasible set $\mathcal{U}_t$ or a ***cyclic*** feasible sets $\{\mathcal{U}_{t_1}, \mathcal{U}_{t_2}, \cdots\}$, which each feasible set $\mathcal{U}_t$ consists of those hypothesis vectors which are "compatible" with a subset of the past examples $(x_i,\, y_i)$. In this algorithm, there are two strategies for the non-cyclic case and cyclic case. The non-cyclic strategy projects the current hypothesis $\boldsymbol{w}_t$ to a single affine set $af(U) \coloneqq \cap_{i\in U}\{\boldsymbol{u} \in \mathcal{H} :\, \langle \boldsymbol{u},\, \boldsymbol{x}_i\rangle = y_i\}$ that is determined by the example indices in U_t. The cyclic strategy projects on a sequence of half-spaces $hs(U) \coloneqq \cap_{i\in U}\{\boldsymbol{u} \in \mathcal{H} :\, y_i\,\langle \boldsymbol{u},\, \boldsymbol{x}_i\rangle \geq 1\}$ determined by the example indices in U_t. The boolean ***aggressive*** controls whether an update is made on every trial (aggressive) or only on those trials when a mistake occurs (non-aggressive).

Theorem 3.2. *If* $\{(\boldsymbol{x}_i,\, y_i)\}_{i=1}^{l} \subseteq \mathcal{H} \times \{-1,\, +1\}$ *is a sequence of examples,* $\boldsymbol{w}_1 \in \mathcal{H}$ *a start vector and* M *the set of trials in which Algorithm 3.5 predicted incorrectly, then the cumulative number of mistakes* $|M|$ *of the algorithm is bounded by*

$$|M| \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 B \tag{3.15}$$

for all $\boldsymbol{u} \in hs(\{1, \cdots, l\})$ *with cyclic updating and for all* $\boldsymbol{u} \in af(\{1, \cdots, l\})$ *with non-cyclic updating, where*

$$B = \left(\frac{1}{|M|}\sum_i \left(\|\boldsymbol{x}_i\|^2\right)^{-1}\right)^{-1}. \tag{3.16}$$

Theorem 3.2 bounds the number of mistakes as proportional to the squared norm ("complexity") of the predictor $\boldsymbol{u}$. We can sketch the proof. For instance, let us consider the non-cyclic case. On each trial t there is a projection to a single affine set $\mathcal{U}_t = af(U_t)$. Since we showed that for all $\boldsymbol{u} \in af(\{1, \cdots, l\})$, we have $\sum_{t=1}^{l}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{l+1}\|^2$. If a mistake has occurred at trial t, $y_y = \langle \boldsymbol{w}_{t+1},\, \boldsymbol{x}_t\rangle$, then $1 \leq |\langle \boldsymbol{w}_t,\, \boldsymbol{x}_t\rangle - y_t| \leq |\langle \boldsymbol{w}_t - \boldsymbol{w}_{t+1},\, \boldsymbol{x}_t\rangle - y_t| \leq \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|\,\|\boldsymbol{x}_t\|$. Therefore, $1 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|\,\|\boldsymbol{x}_t\|$,

Algorithm 3.5 Online Projection Algorithm

Input: $\mathcal{D}_\ell = (x_1, y_1), \ldots, (x_\ell, y_\ell) \in \mathcal{H} \times \{-1, +1\}$.
Initialization: $\boldsymbol{w}_1 \in \mathcal{H}$; `cyclic`= ⟨TRUE|FALSE⟩;
`aggressive` = ⟨TRUE|FALSE⟩
for $t = 1, \ldots, \ell$ **do**
  Predict: receive $\boldsymbol{x}_t$
  $\hat{y}_t = \text{sign}(\langle \boldsymbol{w}_t, \mathbf{x}_t \rangle)$
  Update: receive y_t
  if $\hat{y}_t \neq y_t$ **then** $M = M \cup \{t\}$
  U_t = `strategy`$(M, \boldsymbol{w}_t, \mathcal{D}_t)$
  if `cyclic` **then**
    $\boldsymbol{w}_{t_1} = \boldsymbol{w}_t$; $i = 1$
1   **if** $\langle \boldsymbol{w}_{t_i}, \boldsymbol{x}_t \rangle y_t \leq 0$ *or* `aggressive` **then**
      $\boldsymbol{w}_{t_{i+1}} = P(\text{hs}(\{t\}); \boldsymbol{w}_{t_i})$; $i = i + 1$
    while $\exists \tau_i \in U_t : \langle \boldsymbol{w}_{t_i}, \boldsymbol{x}_{\tau_i} \rangle y_{\tau_i} < 0$ **do**
2       $\boldsymbol{w}_{t_{i+1}} = P(\text{hs}(\{\tau_i\}); \boldsymbol{w}_{t_i})$; $i = i + 1$
    end
    $\boldsymbol{w}_{t+1} = \boldsymbol{w}_{t_i}$
  else
    if $\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle y_t \leq 0$ *or* `aggressive` **then**
3       $\boldsymbol{w}_{t+1} = P(\text{af}(U_t); \boldsymbol{w}_t)$
  end
end

which implies $\|\boldsymbol{x}_t\|^{-1} \leq \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|$. Hence, $\sum_{t \in M} \|\boldsymbol{x}_t\|^{-2} \leq \sum_{t=1}^{l} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2$. Combining them, we have $\sum_{t \in M} \|\boldsymbol{x}_t\|^{-2} \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 + \|\boldsymbol{u} - \boldsymbol{w}_{l+1}\|^2$, which gives us $|M| \leq \left(\|\boldsymbol{u} - \boldsymbol{w}_t\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{l+1}\|^2\right) B$. After dropping the term $\|\boldsymbol{u} - \boldsymbol{w}_{l+1}\|^2$, we then get formula 3.15. The argument for the cyclic case follows the above argument except that the left hand side of the analogous inequality to $\sum_{t=1}^{l} \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{l+1}\|^2$ contains sub-terms due to repeatedly cycling through the past examples, which is line 2 in algorithm 3.5. These terms may all be lower-bounded by zero except the term corresponding to the first projection, which is line 1 in algorithm 3.5.

3.3 Online Learning over Graphs

As shown in section 3.1, the reproducing kernel of the Hilbert space defined over the graph is given by the pseudo-inverse of the Laplacian matrix, $\boldsymbol{L}^+$. This suggests that we can apply the above projection algorithm, with its associated guarantees, by letting the input vector $x_t$be replaced by the t^{th} row of $\boldsymbol{L}^+$. We saw in the previous section that the number of cumulative mistakes is upper bounded by the product of two terms: one involving $\|\boldsymbol{u} - \boldsymbol{w}_1\|^2$, and the other involving the harmonic mean of the input patterns that were misclassified. Since the initial weight vector $\boldsymbol{w}$ is arbitrary, we will consider the first term to be bounded more generally by $\|\boldsymbol{u}\|^2$. In the following analysis, we will derive upper bounds for each of these terms using the properties of our graph, and of the labellings given to us.

Theorem 3.3. *Let $\boldsymbol{g}^+$ be the positively labelled vertices, and let $\boldsymbol{g}^-$ be the negatively labelled vertices. Let $|\boldsymbol{g}^+| = n^+$, $|\boldsymbol{g}^-| = n^-$, and assume WLOG that $n^+ >= n^-$. We claim that $\|\boldsymbol{u}\|^2 \leq \partial(\boldsymbol{g}^+, \boldsymbol{g}^-)(1 + \frac{n_+}{n_-})^2$*

Proof. To show this, we consider the vector $\boldsymbol{f}$, with $f_i = 1$ if $g_i = 1$, and $f_i = -\frac{n^+}{n^-}$ if $g_i = -1$. Now suppose we order our vertices so that the first n^+ vertices are those in $\boldsymbol{g}^+$, while the last n^+ vertices are those in $\boldsymbol{g}^-$. Now consider the Laplacian $\boldsymbol{L}$ of this matrix with the vertices ordered in such a way. We are interested in the sums of the entries in the off-diagonal blocks, the sums of whose entries are each the negative of the number of intra-partition edges. We will denote this sum by S, as opposed to the sub-matrix $\boldsymbol{S}$.

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{A} & -\boldsymbol{S} \\ -\boldsymbol{S}^T & \boldsymbol{B} \end{bmatrix} \quad . \tag{3.17}$$

Note that the sum of entries in the left partition of the matrix for any given row is always the negative of

the sum to the right of the partition. Using this and the fact that the sum of entries in both $\boldsymbol{S}$ and $\boldsymbol{S}^T$ are both equal to $\boldsymbol{S}$, we find after some algebra that

$$\|\boldsymbol{f}\|^2 = \langle \boldsymbol{f}, \boldsymbol{f} \rangle = \boldsymbol{f}^T \boldsymbol{L} \boldsymbol{f} = \left(1 + \frac{n^+}{n^-}\right)^2 \boldsymbol{S} \tag{3.18}$$

□

Finally, combined with the bound on the power mean of the input patterns in terms of the graph diameter, we achieve a bound on the number of mistakes.

As before, our weight vector $\boldsymbol{g}$ reoresents a real-valued function on our Hilbert space. But in the graph setting, this Hilbert space and its reproducing kernel $\boldsymbol{L}^+$and the vector $\boldsymbol{g}$ admit a specific interpretation.

We can get more intuition about the basic objects such as $\|\boldsymbol{g}\|$ and $\boldsymbol{L}^+$ by using the concept of conductance (it is no coincidence that the continuous version of the Laplacian appears in the heat equation). If we think of f as some distribution over the nodes, we can think of $\boldsymbol{L}\boldsymbol{f}$ roughly as the flux induced at each of the nodes by that distribution over the graph. Then the form $\boldsymbol{g}^T \boldsymbol{L} \boldsymbol{f}$ represents some weighted measurement of this flux with the weights given by $\boldsymbol{g}$. For example, if f is a binary vector representing the boundary of some open set on the graph $\langle \boldsymbol{f}, \boldsymbol{g} \rangle$ would be the flux produced by $\boldsymbol{g}$ as measured on the boundary represented by $\boldsymbol{f}$. We can even use this formalism to quickly prove a version of Stokes' Theorem on graphs (we know Stokes' Theorem holds on general smooth manifolds, so this just a particularly simple proof of a special case).

Let $G(V, E, W)$ be a graph imbued with the topology generated by the open neighbourhoods $N_i = \{v_j : (i, j) \in E\}$. Suppose we have some subset $U(G)$ of vertices set $V(G)$, i.e. $U(G) \subset V(G)$. Then we can define the boundary of $U(G)$ as the set of all vertices contained in the closure of $U(G)$ but not contained in $U(G)$ itself $\partial U(G) = \{v_j : v_j \in \bar{U}(G) - U(G)\}$. Now suppose our vertices $V(G)$ are ordered such that $v_1, \cdots, v_k \in U(G)$, $v_{k+1}, \cdots, v_l \in \partial U(G)$, and $v_{l+1}, \cdots, v_n \in V(G) - \bar{U}(G)$. Now let $\mathbf{1}_U$ be a binary vector with all 1 for the first k entries, and 0 elsewhere, and let $\mathbf{1}_{\partial U}$ be the binary vector with all 1 for entries $k+1$ through l, and 0 elsewhere. Then according to the above interpretation of the norm,

$$\boldsymbol{F}_U \coloneqq \mathbf{1}_U \boldsymbol{L} \boldsymbol{g} \tag{3.19}$$

is the total flux induced by distribution g over the interior of $U(G)$, while

$$\boldsymbol{F}_{\partial U} \coloneqq \mathbf{1}_{\partial U} \boldsymbol{L} \boldsymbol{g} \tag{3.20}$$

is the total flux induced by $\boldsymbol{g}$ as measured over the boundary of $U(G)$. Our claim is that

$$\boldsymbol{F}_U = -\boldsymbol{F}_{\partial U}. \tag{3.21}$$

We now give an informal proof. Let us denote by d_i the degree of vertex i, d_i^{int} by the number of edges connecting vertex i to points in $U(G)$, and d_i^{ext} by the number of edges connecting vertex i to points not in $U(G)$, so that

$$d_i = d_i^{int} + d_i^{ext}. \tag{3.22}$$

In particular, for any $v_i \in U$, d_i^{ext} is the number of edges connecting v_i to points in $\partial U(G)$, since by definition all other vertices are not connected to v_i. Then for the left hand side of formula 3.21, note that the only contribution to the i^{th} entry is given by the first k columns of $\boldsymbol{L}$, and a straightforward calculation shows that

$$\mathbf{1}_U \boldsymbol{L} \boldsymbol{g} = \sum_{i \in U} (d_i - d_i^{int}) = \sum_{i \in U} d^{ext}. \tag{3.23}$$

For the right hand side of formula 3.21, we can see that the only contribution to the sum is from entries $k+1$ to l, which another straightforward calculation shows that

$$\mathbf{1}_{\partial U} \boldsymbol{L} \boldsymbol{g} = \sum_{i \in \partial U} d_i^{ext}. \tag{3.24}$$

Thus

$$\mathbf{1}_U \boldsymbol{L}\boldsymbol{g} = -\mathbf{1}_{\partial U} \boldsymbol{L}\boldsymbol{g}, \qquad \forall \boldsymbol{g} \in H. \tag{3.25}$$

Despite the simplicity of the proof and the fact that there is a very general form of Stoke's theorem over smooth manifolds, this graphical version of Stokes' theorem has not been seen by the authors, and could potentially be leveraged for new graphical algorithms.

To gain intuition about L^+, we turn back to the representation theorem, namely that $\boldsymbol{f}(v_i) = \boldsymbol{L}_i^+ \boldsymbol{L}\boldsymbol{f}$. In the heat analogy, this could be interpreted as saying that the heat at each vertex can be expressed in terms of, or derived from, the flux through every vertex. So in this heat analogy, $\boldsymbol{L}$ sends a heat distribution $\boldsymbol{f}$ over each node to a flux through each vertex. Conversely, $\boldsymbol{L}_i^+$ sends some definition of fluxes over the graph back to some heat distribution that would have induced it. This relationship is shown in figure 3.1. Intuitively,

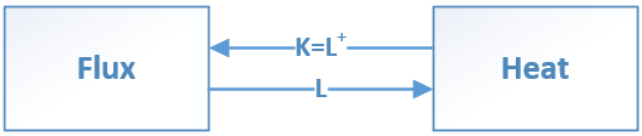

Figure 3.1: Bidirectional Mapping Between Heat and Flux

if the graph is disconnected this distribution should not be unique, since a constant added to the distribution in each of the connected components will not affect the flux induced through the graph. Mathematically, this is affirmed by noting that

$$\boldsymbol{L}^+ = \boldsymbol{L}^{-1} \tag{3.26}$$

if and only if all eigenvalues of L are non-zero which, is exactly when the graph is connected.

3.4 Drawbacks with the above Approach

Large diameter gaphs

With that in mind, we can introduce the mathematical framework for the online projection algorithm of Herbster, Pontil and Wainer (2005). Herbster seeks to minimize

-projection description (tom)

-learning bounds (Paul)

Extension of the herbster paper - use different laplacians simulataneously by considering them each as different kernels in some set of kernels, and then applying the objective function for multiple kernels (need to flesh out this idea)

Bibliography

Bapat, Ravindra B. (2014). *Graphs and Matrices*. London, UK: Springer-Verlag. 193 pp. ISBN: 9781447165682.

Belkin, Mikhail, Irina Matveeva and Partha Niyogi (2004). *Regularization and Semi-supervised Learning on Large Graphs*. Ed. by John Shawe-Taylor and Yoram Singer. Learning Theory 17th Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004. Proceedings. Berlin, Deutschland: Springer Science+Business Media. 654 pp. ISBN: 9783540222828 (cit. on pp. 5, 6).

Bengio, Yoshua, Olivier Delalleau and Nicolas Le Roux (2006). *Label Propagation and Quadratic Criterion*. Ed. by Olivier Chapelle, Bernhard Schölkopf and Alexander Zien. Semi-Supervised Learning. Cambridge, Massachusetts, USA: The MIT Press. 528 pp. ISBN: 9780262033589 (cit. on pp. 4, 7).

Bousquet, Olivier and André Elisseeff (2001). *Algorithmic Stability and Generalization Performance*. Ed. by Todd K. Leen, Thomas G. Dietterich and Volker Tresp. Advances in Neural Information Processing Systems 13 Proceedings of the 2000 Conference. Cambridge, MA, USA: The MIT Press. 1128 pp. ISBN: 9780262122412 (cit. on p. 6).

Herbster, Mark, Massimiliano Pontil and Lisa Wainer (2005). *Online Learning over Graphs*. Ed. by Luc De Raedt and Stefan Wrobel. ICML '05 Proceedings of the 22nd International Conference on Machine Learning. New York, NY, USA: Association for Computing Machinery. 1113 pp. ISBN: 1595931805 (cit. on pp. 7, 9, 10, 13).

Mohri, Mehryar, Afshin Rostamizadeh and Ameet Talwalkar (2012). *Foundations of Machine Learning*. Cambridge, Massachusetts, USA: The MIT Press. 432 pp. ISBN: 9780262018258 (cit. on p. 6).

Zhou, Dengyong, Olivier Bousquet, Thomas Navin Lal, Jason Weston and Bernhard Schölkopf (2004). *Learning with Local and Global Consistency*. Ed. by Sebastian Thrun, Lawrence K. Saul and Bernhard Schölkopf. Advances in Neural Information Processing Systems 16 Proceedings of the 2003 Conference. Cambridge, Massachusetts, USA: The MIT Press. 1728 pp. ISBN: 9780262201520 (cit. on p. 7).

Zhu, Xiaojin and Zoubin Ghahramani (2002). "Learning from Labelled and Unlabelled Data with Label Propagation". In: *Technical Report CMU-CALD-02-107, Carnegie Mellon University*.