Online Learning over Graphs

Mark Herbster M.HERBSTER@CS.UCL.AC.UK
Massimiliano Pontil M.PONTIL@CS.UCL.AC.UK
Lisa Wainer L.WAINER@CS.UCL.AC.UK
Department of Computer Science, University College London, Malet Place, London WC1E 6BT, UK

Abstract

We apply classic online learning techniques similar to the perceptron algorithm to the problem of learning a function defined on a graph. The benefit of our approach includes simple algorithms and performance guarantees that we naturally interpret in terms of structural properties of the graph, such as the algebraic connectivity or the diameter of the graph. We also discuss how these methods can be modified to allow active learning on a graph. We present preliminary experiments with encouraging results.

1. Introduction

We consider online learning over a graph. In our online learning model an adversary presents a sequence of (pattern, label) pairs over a series of trials. In each trial the learner is first presented with a pattern or object, the learner then predicts the label and finally receives the true label either making a mistake or not. The goal is to minimize the number of mistakes (cumulative error) on the sequence. Here, the "patterns" are identified as the vertices of a graph G which may be either provided by nature or may have been derived through some similarity metric on the objects (Belkin & Niyogi, 2004; Zhu et al., 2003a). For example, G could be a social network of people and we may wish to predict peoples' preferences for products; protein families can be associated with a graph structure and we may wish to predict interactive properties of individual proteins. In these applications, the data (labeled vertices) do not need to be i.i.d. as typically assumed in statistical learning. Further benefits of our approach include simple training algorithms and performance guarantees that we naturally interpret in terms of structural properties of the graph.

Learning a function defined on a graph G from a set of labeled vertices has recently received considerable attention in machine learning. This set-up is often referred to as semi-supervised learning on a graph. It has been studied and motivated from different perspectives in (Belkin & Niyogi, 2004; Kondor & Lafferty, 2002; Zhu et al., 2003a; Smola & Kondor, 2003; Belkin et al., 2004). A common theme of these papers is that they represent functions defined on the graph by a Hilbert space associated with the graph Laplacian. We review these ideas in Section 2. We then present a family of online learning algorithms in an abstract Hilbert space in Section 3. We apply these algorithms to the graph setting in Section 4, and interpret their bounds in terms of the structural properties of the graph. In Section 5 we turn our attention to the problem of online active learning on a graph and present an algorithm for this purpose. Finally, in Section 6 we report about preliminary experiments with our methods.

2. Hilbert Space of Functions on a Graph

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$, edge set $E(G) \equiv E \subseteq \{(i, j) : i < j\}_{i,j \in V}$ and $n \times n$ adjacency matrix $\mathbf{A} = (A_{ij} : i, j \in V)$ such that $A_{ij} = A_{ji} = 1$ if $(i, j) \in E$ and zero otherwise[1]. The graph Laplacian $\mathbf{L}$ is the $n \times n$ matrix defined as $\mathbf{L} := \mathbf{D} - \mathbf{A}$, where $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ and d_i is the degree of vertex i.

Let $\mathcal{R}(G)$ be the linear space of real-valued functions defined on the graph, i.e., an $n-$dimensional vector space whose elements are the real vectors $\mathbf{g} = (g_1, \ldots, g_n)^\top$, where "$^\top$" denotes transposition. On $\mathcal{R}(G)$ we introduce the semi-inner product

$$\langle \mathbf{f}, \mathbf{g} \rangle := \mathbf{f}^\top \mathbf{L} \mathbf{g}.$$

The function $\langle \cdot, \cdot \rangle$ is well defined since $\mathbf{L}$ is symmet-

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

[1]The ideas we discuss below naturally extend to weighted graphs as well.

ric and positive semidefinite. Moreover, the function $\|\mathbf{g}\| := \sqrt{\langle \mathbf{g}, \mathbf{g} \rangle}$, $\mathbf{g} \in \mathcal{R}(G)$ is a semi-norm since $\|\mathbf{g}\| = 0$ if $\mathbf{g}$ is a constant vector. This fact can be easily verified by noting that

$$\|\boldsymbol{g}\|^2 = \sum_{(i,j)\in E(G)} (g_i - g_j)^2 \ . \tag{1}$$

The semi-norm $\|\cdot\|$ is a measure of smoothness ("complexity"). Thus if $\|\mathbf{g}\|$ is small then $\mathbf{g}$ varies slowly on the graph; i.e., if $(i,j) \in E$ then $g_i \approx g_j$.

Recall that G has r connected components if and only if $\mathbf{L}$ has r eigenvectors with zero eigenvalues. Those eigenvectors are piece-wise constant on the connected components of the graph. In particular, G is connected if and only if the constant vector is the only eigenvector of $\mathbf{L}$ with zero eigenvalue. See, e.g., (Chung, 1997). Let $\{\lambda_i, \boldsymbol{u}_i\}_{i=1}^n$ be a system of eigen-values/vectors of $\mathbf{L}$ where $\lambda_1 = \cdots = \lambda_r = 0$ and $0 < \lambda_{r+1} \leq \cdots \leq \lambda_n$ and define the linear subspace $\mathcal{H}(G)$ of $\mathcal{R}(G)$ which is orthogonal to the eigenvectors with zero eigenvalue, that is,

$$\mathcal{H}(G) := \{\mathbf{g} : \mathbf{g}^\top \boldsymbol{u}_i = 0,\ i = 1, \ldots, r\}. \tag{2}$$

Clearly, the restriction of the semi-norm $\|\cdot\|$ on $\mathcal{H}(G)$ is a norm. What is the reproducing kernel of $\mathcal{H}(G)$? According to, e.g., (Wahba, 1990) this is an $n \times n$ symmetric matrix $\mathbf{K}$ such that, for every $\mathbf{g} \in \mathcal{H}(G)$ and $i \in V$ the reproducing kernel property

$$g_i = \langle \mathbf{K}_i, \mathbf{g} \rangle \tag{3}$$

holds, where $\mathbf{K}_i$ is the i-th column of $\mathbf{K}$. We claim that $\mathbf{K} = \mathbf{L}^+ = \sum_{i=r+1}^n \lambda_i^{-1} \boldsymbol{u}_i \boldsymbol{u}_i^\top$, where "+" denotes pseudoinverse. Indeed, $\mathbf{K}$ is symmetric and a direct computation gives $\mathbf{L}\mathbf{L}^+ = \mathbf{I} - \sum_{i=1}^r \boldsymbol{u}_i \boldsymbol{u}_i^\top$. Consequently, if $\mathbf{g} \in \mathcal{H}(G)$ then $\mathbf{L}\mathbf{L}^+\mathbf{g} = \mathbf{g}$ which implies that $g_i = \boldsymbol{e}_i \mathbf{L}^+ \mathbf{L}\mathbf{g} = \mathbf{K}_i \mathbf{L}\mathbf{g} = \langle \mathbf{K}_i, \mathbf{g} \rangle$, where $\boldsymbol{e}_i$ is the $i-$th coordinate vector.

Within this framework, we wish to learn *online* a classification function $\mathbf{g} \in \mathcal{H}(G)$ on the basis of a set of labeled vertices. Without loss of generality we assume that the first $\ell \leq n$ vertices are labeled and let $y_1, ..., y_\ell \in \{-1, 1\}$ be the corresponding labels. Our ideas elaborate on previous batch algorithms for learning a function on a graph. In particular, in (Belkin & Niyogi, 2004) the function $\mathbf{g}$ is computed as the minimizer of a regularized least-square error while in (Zhu et al., 2003a) $\mathbf{g}$ is (essentially) obtained as the minimal norm interpolant in $\mathcal{H}(G)$ to the labeled vertices, i.e., the unique solution to the problem

$$\min_{\mathbf{g}\in\mathcal{H}(G)} \{\|\mathbf{g}\| : g_i = y_i, i = 1, \ldots, \ell\}. \tag{4}$$

These methods compute $\mathbf{g}$ by solving a squared linear system of n and $n - \ell$ equations respectively. On the contrary, in this paper we use the representer theorem, see, e.g. (Wahba, 1990), to express $\mathbf{g}$ as

$$g_i = \sum_{j=1}^{\ell} K_{ij} c_j.$$

This approach is more advantageous if the kernel $\mathbf{K}$ can be computed off-line. Indeed, the computation of the parameter vector $\mathbf{c} = (c_1, \ldots, c_\ell)$ involves solving a linear system of ℓ equations and, typically, $\ell \ll n$. In particular, the solution of (4) is obtained as $\mathbf{c} = \widetilde{\mathbf{K}}^+ \mathbf{y}$ where $\widetilde{\mathbf{K}} = (K_{ij})_{i,j=1}^{\ell}$.

3. Projection Algorithms

Let $\mathcal{H}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle$ and norm $\|\cdot\| := \sqrt{\langle \cdot, \cdot \rangle}$, e.g., the above Hilbert space $\mathcal{H}(G)$ of functions on a graph G.

We consider the following well-known online learning model. Learning proceeds in trials $t = 1, 2, \ldots, \ell$. The algorithm maintains a parameter vector (hypothesis), denoted by $\boldsymbol{w}_t \in \mathcal{H}$. In each trial the algorithm receives a *pattern* $\mathbf{x}_t \in \mathcal{H}$. It then produces some action or prediction denoted by $\hat{y}_t$, a function of $\mathbf{x}_t$ and $\boldsymbol{w}_t$. Finally, the algorithm receives a *label* y_t and incurs a loss $L(y_t, \hat{y}_t)$ measuring the discrepancy between y_t and $\hat{y}_t$. For simplicity, we only consider binary classification, i.e., the labels are in $\{-1, 1\}$, $\hat{y}_t = \text{sign}(\langle \boldsymbol{w}_t, \mathbf{x}_t \rangle)$ and we measure discrepancy by counting mistakes. We provide a bound on the cumulative mistakes in terms of any member of the *realizable* set of predictors, i.e., the set of vectors $\boldsymbol{u}$ that separate the data with a given minimum margin.

Our main algorithm is similar to the well-known mistake-driven perceptron algorithm; but it is designed to have provable bounds even with an *aggressive* updating strategy (see below). The algorithm was directly inspired by (Herbster, 2001) which is broadly generalized in (Shalev-Shwartz et al., 2004).

The key tool which we use to repeatedly construct updates in the algorithms below is *projection*.

Definition 3.1. *The projection of a point $\boldsymbol{w} \in \mathcal{H}$ onto a closed convex nonempty set $\mathcal{N} \subseteq \mathcal{H}$ is defined by*

$$P(\mathcal{N}; \boldsymbol{w}) := \arg\min_{\boldsymbol{u}\in\mathcal{N}} \|\boldsymbol{u} - \boldsymbol{w}\|. \tag{5}$$

The following version of the Pythagorean Theorem is well-known and provides the main tool we use to study Algorithm 2 below.

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

Algorithm 1: Prototypical projection algorithm.

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*

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

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

In the next lemma we summarize some further facts about projections. We first introduce some notation. We let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{\ell} \subseteq \mathcal{X} \times \{-1, 1\}$ be an example set; and let $\mathcal{D}_t$ denote the first t examples of $\mathcal{D}$. For every $U \subseteq \{1, \ldots, \ell\}$ we define the sets $\mathrm{hs}(U) := \bigcap_{i \in U} \{\boldsymbol{u} \in \mathcal{H} : y_i \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle \geq 1\}$ and $\mathrm{af}(U) := \bigcap_{i \in U} \{\boldsymbol{u} \in \mathcal{H} : \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle = y_i\}$. Finally, a classifier $\boldsymbol{u}$ is said *unit-separating* for examples $\mathcal{D}$ if $\boldsymbol{u} \in \mathrm{hs}(\{1, \ldots, \ell\})$.

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

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

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

Von Neumann first proved that a sequence of projections between two closed convex subsets converge to a point in their intersection. Since then this idea has been broadly applied within the optimization community and it is known as the method of alternating projections, see (Bauschke & Borwein, 1996) for a review.

The following lemma shows the convergence of the prototypical projection algorithm (Algorithm 1).

Lemma 3.2. *Let $\{\mathcal{U}_1, \ldots, \mathcal{U}_\ell\} \subseteq \mathcal{H}$ be a sequence of convex sets and $\boldsymbol{w}_1$ a start vector for Algorithm 1. Then for every $\boldsymbol{u} \in \bigcap_{t=1}^{\ell} \mathcal{U}_t$, we have that*

$$\sum_{t=1}^{\ell} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2. \quad (6)$$

Proof. On any trial $t = 1, \ldots, \ell$ the Theorem 3.1 implies, for all $\boldsymbol{u} \in \mathcal{U}_t$, that $\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \|\boldsymbol{u} - \boldsymbol{w}_t\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|^2$. The result follows by summing this inequality over all trials. □

We use the prototypical projection algorithm to produce our main algorithm (see Algorithm 2 below) by associating the *feasible set* sequence $\{\mathcal{U}_1, \ldots, \mathcal{U}_\ell\}$ with the example sequence $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\ell, y_\ell)\}$. Each feasible set $\mathcal{U}_t$ consists of those hypothesis vectors which are "compatible" with a subset of the past examples.

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 $\boldsymbol{x}_t$
  $\hat{y}_t = \mathrm{sign}(\langle \boldsymbol{w}_t, \mathbf{x}_t \rangle)$
  Update: receive y_t
  if $\hat{y}_t \neq y_t$ **then** $M = M \cup \{t\}$
  $U_t = \texttt{strategy}(M, \boldsymbol{w}_t, \mathcal{D}_t)$
  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**
      $\boldsymbol{w}_{t_{i+1}} = P(\mathrm{hs}(\{t\}); \boldsymbol{w}_{t_i})$; $i = i + 1$
    while $\exists \tau_i \in U_t : \langle \boldsymbol{w}_{t_i}, \boldsymbol{x}_{\tau_i} \rangle y_{\tau_i} < 0$ **do**
2       $\boldsymbol{w}_{t_{i+1}} = P(\mathrm{hs}(\{\tau_i\}); \boldsymbol{w}_{t_i})$; $i = i + 1$
    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       $\boldsymbol{w}_{t+1} = P(\mathrm{af}(U_t); \boldsymbol{w}_t)$
  end
end

Algorithm 2: Online projections

The algorithm is parameterized by a `strategy` function, two boolean variables, `cyclic` and `aggressive`, and a start vector $\boldsymbol{w}_1$. The strategy function on trial t determines the index set U_t. This in turn determines the feasible set (noncyclic) $\mathcal{U}_t$ or a sequence (cyclic) of feasible sets $\{\mathcal{U}_{t_1}, \mathcal{U}_{t_2}, \ldots\}$. We assume that the example t is included in the set U_t for every t. Such a strategy is called *corrective*. Typical strategies include the simple perceptron-like strategy ($U_t = \{t\}$), minimum norm interpolation ($U_t = \{1, \ldots, t\}$) when noncyclic, or when cyclic this is analogous to a perceptron where on each trial we cycle through past examples which are currently "predicting incorrectly." The noncyclic strategy projects the current hypothesis $\boldsymbol{w}_t$ to a single affine set $\mathcal{U}_t$ determined by the example indices in U_t whereas the cyclic one projects on a sequence of halfspaces 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). In the active learning setting, we will reap a benefit from the aggressive versions of the algorithm.

The following theorem bounds the number of mistakes as proportional to the squared norm ("complexity") of the predictor $\boldsymbol{u}$. In an adversarial setting a mistake may be forced on every trial. However, if one additionally assumes that the predictor has small squared

norm, the total mistakes are hence strictly bounded. Recall that the p-power mean of a set of nonnegative numbers $\{a_i\}_{i=1}^n$ is defined as

$$\mu(p;\{a_i\}_{i=1}^n) = \left(\frac{1}{n}\sum {a_i}^p\right)^{\frac{1}{p}} \tag{7}$$

if $p \in \mathbb{R}$ and appropriately in the limit when $p \in \{-\infty, 0, \infty\}$. In particular, $\mu(\infty;\{a_i\}) = \max\{a_i\}$.

Theorem 3.2. *If $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{\ell} \subseteq \mathcal{H} \times \{-1,1\}$ is a sequence of examples, $\boldsymbol{w}_1 \in \mathcal{H}$ a start vector and M the set of trials in which Algorithm 2 predicted incorrectly, then the cumulative number of mistakes $|M|$ of the algorithm is bounded by*

$$|M| \le \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 B \tag{8}$$

for all $\boldsymbol{u} \in \mathrm{hs}(\{1,\ldots,\ell\})$ with cyclic updating and for all $\boldsymbol{u} \in \mathrm{af}(\{1,\ldots,\ell\})$ with noncyclic updating, where

$$B = \mu(-1;\{\|\boldsymbol{x}_i\|^2\}_{i\in M}). \tag{9}$$

Proof. The algorithm projects onto a sequence of convex sets determined by each index set U_t. In the noncyclic case, on each trial t there is a projection to a single affine set $\mathcal{U}_t = \mathrm{af}(U_t)$, while in the cyclic case on each trial there is potentially a sequence of projections to halfspaces $\{\mathrm{hs}(\{\tau_1\}), \mathrm{hs}(\{\tau_2\}), \ldots\}$ where each $\tau_i \in U_t$. We now argue in detail the noncyclic case and sketch the proof for the cyclic case.

By Lemma 3.2 we have, for all $\boldsymbol{u} \in \mathrm{af}(\{1,\ldots,\ell\})$, that

$$\sum_{t=1}^{\ell} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2 \le \|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2. \tag{10}$$

If a mistake has occurred at trial t we have that

$$1 \le |\langle \boldsymbol{w}_t, \mathbf{x}_t\rangle - y_t| \le |\langle \boldsymbol{w}_t - \boldsymbol{w}_{t+1}, \mathbf{x}_t\rangle| \le \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|\|\mathbf{x}_t\|$$

where the first two inequalities follow from the fact that there has been a mistake and that the `strategy` function is corrective, i.e., $y_t = \langle \boldsymbol{w}_{t+1}, \mathbf{x}_t\rangle$, and the final inequality follows by the Cauchy-Schwarz inequality. Consequently we have, for all $t \in M$, that $\|\mathbf{x}_t\|^{-1} \le \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|$, which implies that $\sum_{t\in M} \|\mathbf{x}_t\|^{-2} \le \sum_{t=1}^{\ell} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2$. Combining the last inequality with inequality (10) we have that

$$|M| \le \left(\|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - \|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2\right)\mu(-1;\{\|\boldsymbol{x}_i\|^2\}_{i\in M})$$

which after dropping the term $\|\boldsymbol{u} - \boldsymbol{w}_{\ell+1}\|^2$ gives (8) with B as in (9).

The argument for the cyclic case follows the above argument except that the left hand side of the analogous inequality to (10) contains sub-terms due to repeatedly cycling through the past examples (line 2 in Algorithm 2); these terms may all be lower-bounded by zero except the term corresponding to the first projection (line 1 in Algorithm 2). □

The previous bound requires that the data is realizable, i.e., separable for the cyclic case and that an exact interpolation of the labels exists in the noncyclic case. This requirement is not particularly onerous for the graph learning problem. Given a connected n-vertex graph, the corresponding Hilbert space $\mathcal{H}(G)$ in (2) is $n-1$ dimensional. Hence, if the data sequence does not contain any duplicate patterns and is of length less than n it is always realizable. Furthermore, the bound for the cyclic case is always better than the bound for the noncyclic case since $\mathrm{af}(U) \subset \mathrm{hs}(U)$ for every $U \subseteq \{1,\ldots,\ell\}$.

Recall that for every $r, s \in \mathbb{R} \cup \{-\infty, \infty\}$ with $r \le s$, the power mean inequality is

$$\mu(r;\{a_i\}) \le \mu(s;\{a_i\}). \tag{11}$$

Using this inequality we can further bound the constant B in (9) as $B \le \mu(1;\{\|\boldsymbol{x}_i\|^2\}_{i\in M}) \le \max_{1\le t\le \ell} \|\boldsymbol{x}_t\|^2$. Note that for this choice of B, the bound (8) is equivalent to the margin bound proved for the perceptron. Theorem 3.2 refines the upper bound on B to the harmonic mean of the squared norm of the misclassified patterns. The refinement of B to the arithmetic mean was previously given in (Gentile & Warmuth, 1998).

4. Application to the Graph Setting

The analysis in Section 3 provides an algorithm and a general mistake bound in an abstract Hilbert space $\mathcal{H}$. We apply these results to graphs by setting $\mathcal{H} = \mathcal{H}(G)$. We proceed to discuss the implementation and then to discuss our bounds in terms of the structural properties of the graph.

In order to run our online algorithm in the space $\mathcal{H}(G)$ we proceed as follows. We first compute the kernel $\mathbf{K} = \mathbf{L}^+$. This requires $O(n^3)$ computations but has the advantage that multiple problems may be run with the same graph kernel. Then, to run Algorithm 2 we simply replace pattern $\boldsymbol{x}_t$ by $\mathbf{K}_t$, identify $\boldsymbol{w}_t$ by $\mathbf{g}_t$ and use the reproducing kernel property to compute $\langle \mathbf{g}_t, \mathbf{K}_t\rangle$ at each iteration. This version of the projection algorithm is similar to the "kernel perceptron," see (Freund & Schapire, 1999). The computational complexity of the algorithm depends on the function `strategy` and the choice of the parameters `cyclic` and `aggressive`. For example, when noncyclic, aggressive and $U_t = \{1,\ldots,t\}$ (minimal norm interpolation) it requires $O(t^2)$ computations on trial t. However, when cyclic, nonaggressive and $U_t = \{t\}$ it requires

$O(m)$ computations on each trial where m is the current number of mistakes, see (Herbster et al., 2005) for details.

For general Hilbert spaces in Theorem 3.2 the number of mistakes of the algorithm is upper bounded by the product of two terms: first, the squared norm of the predictor; and second, the harmonic mean of the squared norms of the misclassfied patterns. The next two theorems provide a bound for these terms vis-à-vis graph properties. Specifically, the number of mistakes on a partitioned graph is bounded by the product of three quantities (splitting the first term above in two): first, the number of intra-partition edges; second, a measure of the balance of the partition (Theorem 4.1); third, a natural constant of proportionality which depends on the diameter of the graph and the second smallest eigenvalue of the graph Laplacian, the *algebraic connectivity* of the graph (Theorem 4.2).

A label partition $(\mathbf{g}^+, \mathbf{g}^-)$ of a graph G assigns labels to *each* vertex of the graph; hence $\mathbf{g}^+ = \{i : g_i = 1\}$ and $\mathbf{g}^- = \{i : g_i = -1\}$. The squared norm of a unit-separating classifier is bounded in the following theorem by the optimal partitioning of the graph as constrained by the labelled data.

Theorem 4.1. *If G is a connected graph with a label partition $(\mathbf{g}^+, \mathbf{g}^-)$, $n^+ = |\mathbf{g}^+| > 0, n^- = |\mathbf{g}^-| > 0$ and $\partial(\mathbf{g}^+, \mathbf{g}^-)$ is the number of edges between positive and negative vertices, then there exists a unit-separating classifier $\boldsymbol{u}$ $(\forall i : u_i g_i \geq 1)$ with a norm bounded as*

$$\|\mathbf{u}\|^2 \leq \partial(\mathbf{g}^+, \mathbf{g}^-) \left(\frac{n}{\min(n+, n^-)} \right)^2.$$

Moreover, if $n^+ \geq n-$, we have, for the same classifier, that $\lambda_2 n \frac{n^+}{n^-} \leq \|\boldsymbol{u}\|^2 \leq \lambda_n n \frac{n^+}{n^-}$.

The first bound follows from counting the intra-partition edges while enforcing the constraint $\sum_{i=1}^n u_i = 0$, see (Herbster et al., 2005) for details.

If $p \in V$ we define the eccentricity of vertex p, ρ_p, to be the distance on the graph between p and the furthest vertex on the graph to p, that is,

$$\rho_p = \max_{q \in V} \min |P(p, q)| \leq D_G$$

where the minimum is taken with respect to all paths $P(p, q)$ from p to q and D_G is the diameter of the graph, $D_G = \max_p \rho_p$.

Recall, by the reproducing kernel property (equation (3)), that the squared norm of the pattern of graph vertex p is K_{pp}.

Theorem 4.2. *For connected graph G with Laplacian kernel K and eccentricity function ρ we have that*

$$K_{pp} \leq \min(\frac{1}{\lambda_2}, \rho_p), \quad p \in V. \tag{12}$$

Proof. From the Rayleigh-Ritz characterization of eigenvalues we have, for every $\mathbf{g} \in \mathcal{H}$ that

$$\mathbf{g}^\top L \mathbf{g} \geq \lambda_2 \mathbf{g}^\top \mathbf{g}. \tag{13}$$

If $\boldsymbol{g} = \mathbf{K}_p$ we have $K_{pp} = \boldsymbol{g}^T L \boldsymbol{g} \geq \lambda_2 \boldsymbol{g}^T \boldsymbol{g} \geq \lambda_2 K_{pp}^2$, where the equality follows from equation (3) with $i = p$, the left inequality by (13) and the right inequality by the observation that $g_p = K_{pp}$. By dividing through we have that $K_{pp} \leq \frac{1}{\lambda_2}$ for arbitrary p.

We now show that $K_{pp} \leq \rho_p$. We choose $\mathbf{g} = \mathbf{K}_p$ and note that, since $\mathbf{g} \in \mathcal{H}$ if $g_p = K_{pp} > 0$ then there exists $q \neq p$ such that $g_q < 0$. Indeed, the constant vector has zero eigenvalue and, so, by the definition of $\mathcal{H}(G)$ we have that $\sum_{i=1}^n g_i = 0$. Moreover, since p has eccentricity ρ_p there exists a path $P \subseteq G$ from vertex p to q such that $|E(P)| \leq \rho_p$. Hence, we have that $\sum_{(i,j) \in E(P)} |g_i - g_j| > g_p$. Using equation (11) with $a_{(i,j)} = |g_i - g_j|$, $s = 2$ and $r = 1$ we obtain that

$$\begin{aligned} \sum_{(i,j) \in E(P)} (g_i - g_j)^2 &\geq \frac{\left(\sum_{(i,j) \in E(P)} |g_i - g_j| \right)^2}{|E(P)|} \\ &\geq \frac{\left(\sum_{(i,j) \in E(P)} |g_i - g_j| \right)^2}{\rho_p} \geq \frac{g_p^2}{\rho_p}. \end{aligned}$$

Observe that $\mathbf{K}_p^\top L \mathbf{K}_p = \sum_{(i,j) \in E(G)} (K_{pi} - K_{pj})^2$ by (1) and using (3) we have that

$$g_p = \sum_{(i,j) \in E(G)} (g_i - g_j)^2 \geq \sum_{(i,j) \in E(P)} (g_i - g_j)^2 \geq \frac{g_p^2}{\rho_p}$$

from which, using $g_p = K_{pp}$, the result follows. □

We remark that either $\frac{1}{\lambda_2}$ or the diameter may prove tighter as the bound of a given graph. For example, the ring graph with m vertices has a diameter of $\lceil \frac{m}{2} \rceil$ while $\frac{1}{\lambda_2} = \Theta(m^2)$, but the m-dimensional binary hypercube has a diameter of m but $\frac{1}{\lambda_2} = 2$. We note that (Belkin et al., 2004) has studied the problem of bounding the generalization error of the least square regularized solution in $\mathcal{H}(G)$. Their result also involves λ_2 and the diameter of the graph.

4.1. Complexity of Cyclic Updating

It is interesting to analyze the computational complexity of Algorithm 2 with `cyclic` strategies. This complexity is bounded by the number of updates (executions of lines 1 and 2) to the hypothesis vector. In Section 6 for our experiments we use a cyclic strategy called `C-proj` whose strategy set on trial t is $U_t = \{1, \ldots, t\}$, i.e., the algorithms cycles through the current prefix of the dataset until there are no more mistakes. This strategy is flawed for $\mathbb{R}^n$ in so

far as it is well-known that there exists datasets that require an exponential number of updates in the size of the dataset for the perceptron algorithm to converge; these arguments easily extend to our algorithm. However, the intrinsic geometry of the graph Laplacian assures polynomial convergence in the size of the dataset when there is at least one positive and negative label. Indeed, by Theorem 3.2 there exists a unit-separating hyperplane and, so, in inequality (8) we may bound B by the diameter D_G of the graph to obtain a bound on the number of mistakes. Simple bounds on the number of edges between the positive and negative labels (see Theorem 3.4) lead to a bound of $O(n^3 D_G)$ and when the number of positive and negative labels is balanced of $O(n^2 D_G)$. The diameter D_G is itself bounded by $n-1$. Finally a closer analysis of our algorithm would reveal that the mistake bound also bounds the number of updates for non-aggressive strategies.

5. Active Learning

In this section, we present an active online learning method which builds on our analysis in Section 3. We are given a pool of unlabeled patterns $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$; the *true* outcomes (labels) $\{y_1, \ldots, y_\ell\}$ corresponding to the patterns are unknown and may be chosen adversarially by "nature". The learner has an initial segment of s "free" trials. On each trial the learner may choose a pattern to query from the pool; nature returns a corresponding label; then the learner may ask the next query. After an initial segment of s trials then the standard online learning protocol is followed for $\ell - s$ trials. The goal is to minimize the cumulative error on all *future* non-actively chosen examples.

In the theorem below we slightly generalize the active learning model, i.e., we assume that the set of active trials is any subset A of $\{1, \ldots, \ell\}$ with s elements; this matches the model above when $A := \{1, \ldots, s\}$.

Theorem 5.1. *Given a sequence of examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\ell, y_\ell) \in \mathcal{H} \times \{-1, 1\}$ and a start vector $\boldsymbol{w}_1 \in \mathcal{H}$, let M be the set of trials in which Algorithm 2 predicted incorrectly, A the set of active trials and define the progress Z_A on the set A as*

$$Z_A := \begin{cases} \sum_{t \in A} \|\boldsymbol{w}_t - \boldsymbol{w}_{t+1}\|^2 & \texttt{noncyclic} \\ \sum_{t \in A} \sum_{i=1}^{k_t} \|\boldsymbol{w}_{t_i} - \boldsymbol{w}_{t_{i+1}}\|^2 & \texttt{cyclic} \end{cases}$$

where k_t is number of executions of line 2 of the algorithm at trial t. Then, the number of mistakes of the algorithm during trials $\{1, \ldots, \ell\} \backslash A$ is bounded as

$$|M \backslash A| \le (\|\boldsymbol{u} - \boldsymbol{w}_1\|^2 - Z_A) \mu(-1; \{\|\boldsymbol{x}_i\|^2\}_{i \in M \backslash A}) \quad (14)$$

for all $\boldsymbol{u} \in \mathrm{hs}(\{1, \ldots, \ell\})$ with cyclic updating and for all $\boldsymbol{u} \in \mathrm{af}(\{1, \ldots, \ell\})$ with noncyclic updating.

This theorem modestly refines Theorem 3.2 and motivates our technique for selecting unlabeled examples in active learning. See (Herbster et al., 2005) for a proof.

Equation (14) says that the greater the progress Z_A during the s active trials the fewer mistakes on the remaining trials for a fixed complexity $\|\boldsymbol{u} - \boldsymbol{w}_1\|$. This is justification for using an aggressive algorithm for online active learning as opposed to an unadorned perceptron. When Algorithm 2 is `aggressive` in almost every trial we are guaranteed to make progress (progress is not made at trial t if and only if there are no "margin" errors from examples in U_t.) Thus, a sensible strategy for example selection during the active phase is to *maximize the progress*. For this purpose, we propose a greedy technique which independently chooses at each trial a point x_p such that

$$p = \arg\max_{i \in I} \min_{y \in \{-1,1\}} \|\boldsymbol{w} - P(\{\boldsymbol{u} : \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle y \ge 1\}; \boldsymbol{w})\|^2$$

where $\boldsymbol{w}$ and I are the weight vector and the index set at that trial respectively. This is exactly the maxmin progress on trial t when `cyclic` and $U_t = \{t\}$. In every other case it is a lower bound on the progress. We note that this greedy method is similar to a criteria proposed in (Tong & Koller, 2000) in the context of SVM. Other related works include (Campbell et al., 2000; Warmuth et al., 2003) and references therein. We were also inspired by the work in (Zhu et al., 2003b) which considers active learning on a graph from a probabilistic batch perspective.

Note that the minimum in the above equation equals $(\min(|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle|, 1) - 1)^2 \|\boldsymbol{x}_i\|^{-2}$. In particular, when we apply this equation to a graph G we replace $\boldsymbol{x}_i$ by $\mathbf{K}_i$, $\boldsymbol{w}$ by $\mathbf{g} \in \mathcal{H}(G)$ and choose $I = V$ at every trial. Hence, at each trial we select that vertex p which maximizes over $i \in V$,

$$\frac{(\min(|g_i|, 1) - 1)^2}{K_{ii}} \quad . \quad (15)$$

The quantity (15) suggests that vertices with small K_{ii} are of particular importance. Note that K_{ii} is upper bounded by ρ_i, the minimal path distance to the furthest vertex on the graph. Thus, assuming that K_{ii} scales with ρ_i those vertices with small K_{ii} are more *central to the graph* at a first approximation and, hence, more informative. On the contrary, the numerator in (15) is small if a vertex i is *close to a labeled vertex* since in this case $|g_i| \approx 1$. Hence, our active learning model will query a vertex which optimizes the trade-off between the vertex being as central as possible to the graph and being as far as possible to already labeled vertices. The experiments in the next section confirm this observation. This is also nicely illustrated

Method	Strategy	Aggressive	Cyclic	Selection
`1-proj`	$\{t\}$	false	false	random
`C-proj`	$\{1,\ldots,t\}$	false	true	random
`MNI-ag`	$\{1,\ldots,t\}$	true	false	random
`Act-st`	$\{t\}$	true[2]	false	active 1
`Act-mu`	$\{t\}$	true[2]	false	active 2

Table 1. Different learning methods.

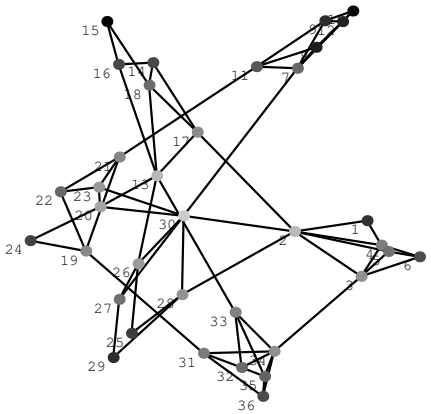

Figure 1. Hier. Random Graph $G_{k-\text{out}}(6;2)$ components grey-scaled K_{pp} : $K_{30,30} = .21$ (min), $K_{15,15} = .94$ (max)

in Figure 1 where even if vertex 15 is maximally uncertain i.e., a margin of $g_{15} = 0$ then criteria (15) will still select vertex 30 preferentially whenever the margin $|g_{30}| \leq 0.51$.

6. Experiments

In this section we present some preliminary experiments with the aim of comparing different instances of the algorithm proposed in this paper. We compare five instances of our algorithm: `1-proj`, `C-proj`, `MNI-ag` (which is similar to the method in (Zhu et al., 2003a)), `Act-st` and `Act-mu`. The first three methods receive data at random while the last two select data actively. Table 1 summarizes the methods' parameters. The last two methods are identical to `1-proj`, except for the way they acquire data: `1-proj` receives data randomly; `Act-st` selects the first t vertices according to criterion (15); `Act-mu` selects the first t vertices based on the minimum margin ($\arg\min_i = |g_i|$), see (Tong & Koller, 2000). We initialize the start vector $\boldsymbol{w}_1 = \mathbf{0}$ for each of the five instances.

We compare the above methods on a digit recognition task and on an artificially generated hierarchical random graph. In the first case we consider the problem of distinguishing the "even" digits from the "odd" digits. The dataset was obtained from the USPS dataset by drawing 100 examples at random from each digit thereby forming a dataset of size 1000. Each digitized image corresponds to a vertex in the graph which we built using 3-NN with the Euclidean distance.

The random graph dataset simulates a noisy multi-cluster concept. It consists of a q–cluster hierarchical random graph where each cluster or sub-graph is a $G_{k-\text{out}}(q;k)$ random graph with q vertices. To generate this sub-graph, for each vertex i we independently sample k of the $q-1$ edges departing from i at random without replacement. We then set $A_{ij} = A_{ji} = 1$ if the edge (i,j) has been sampled at least once in the above process; thus, the resulting graph has at most kq edges and each vertex has degree at least k. We convert this to a two-level hierarchical model as follows. We generate q independent $G_{k-\text{out}}(q;k)$ graphs where each of the q graphs is treated as a meta-vertex in a $G_{k-\text{out}}(q;k)$ graph. We then generate the meta-edges to connect the meta-vertices and realize the resultant meta-edge with a "real" edge by choosing at random a "real" vertex in each meta-vertex. Figure 1 shows an example of a hierarchical $G_{k-\text{out}}(6;2)$ graphs. In our experiments we created hierarchical random graphs with $q = 26$ and $k = 2$. Such a hierarchical graph is labeled by a noisy diffusion process where we initially label 13 randomly chosen vertices as positive and 13 as negative (one vertex per each cluster in the graph), see (Herbster et al., 2005) for more information.

We compare each of the above algorithms on the digit graph (Figure 2 top) and the hierarchical graph (Figure 2 bottom). For this purpose we use the *future* cumulative error to measure performance, that is, $C_{fe}(t) = |M \cap \{t+1,\ldots,\ell\}|$. This is as opposed to plotting the *past* cumulative error, $C_{pe}(t) = |M \cap \{1,\ldots,t\}|$. This is a natural metric for comparing the methods. Indeed, with active learning the first t vertices are actively selected and we wish to measure the performance on the remaining $\ell - t$ vertices which are now received at random. This performance is exactly $C_{fe}(t)$. We also note that for the three non-active methods (`1-perc`, `C-proj` and `MNI-ag`) $C_{fe}(t)$ is the mirror image of $C_{pe}(t)$. Each plot represents a single random graph whose construction was described above. The lines plotted are averaged over 20 random permutations of the data sequence[3].

In the digit graph `C-proj` and `MNI-ag` perform compa-

[2]In order to compare to `1-proj` this algorithm is only aggressive when actively selecting points.

[3]This is literally true with non-active point selection. Each "run" with active selection actually consists of ℓ subruns on each of the ℓ possible prefix-suffix combinations where each suffix is independently randomized.

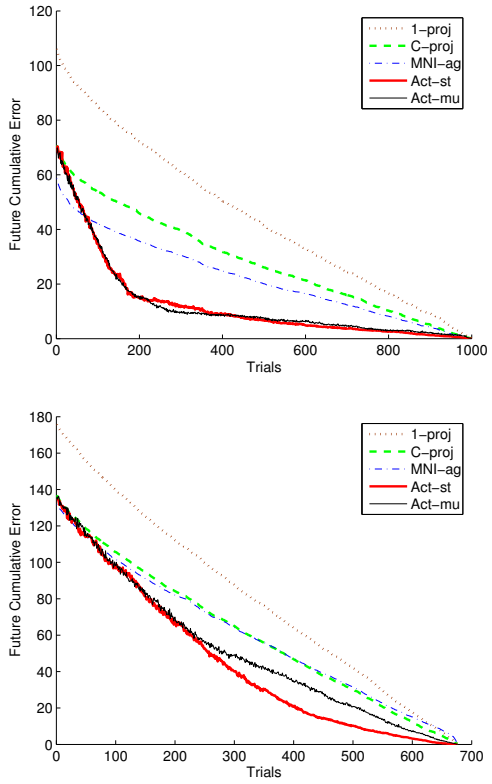

Figure 2. Even vs. Odd (top) and Hierarchical Random Graph experiments.

rably while `1-proj` appears to be weaker. Both active learning methods improved the performance significantly. On the digits graph this effect is dramatically demonstrated by comparing the `1-proj` and `Act-st` future cumulative error from trial 5. These are the same method except that `Act-st` has previously selected 5 points actively via (15) and updated them aggressively while `1-proj` has received them at random, however they lead to future cumulative errors (i.e., the "suffix" for both is a random permutation of the 995 remaining vertices) of 104.4 and 66.9 respectively.

In the random graph experiment, `MNI-ag` and `C-proj` perform similarly and `1-proj` is again the weaker method. Note that this classification task is more difficult than the digit one. In this case it appears that our active method `Act-ag` slightly improves over `Act-mu` when t increases. We speculate that this may be due to the favorable bias of `Act-st` to choose central vertices in the graph (e.g., vertex 30, 2 and 13 in Figure 1) which are especially informative when the task is hard.

Acknowledgments

We thank the anonymous reviewers for valuable comments. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778.

References

Bauschke, H. H., & Borwein, J. M. (1996). On projection algorithms for solving convex feasibility problems. *SIAM Review*, *38*, 367–426.

Belkin, M., Matveeva, I., & Niyogi, P. (2004). Regularization and semi-supervised learning on large graphs. *COLT 2004, Proc.* (pp. 624–638). Springer.

Belkin, M., & Niyogi, P. (2004). Semi-supervised learning on riemannian manifolds. *Machine Learning*, *56*, 209–239.

Campbell, C., Cristianini, N., & Smola, A. (2000). Query learning with large margin classifiers. *ICML 2000, Proc.* (pp. 111–118). Morgan Kaufmann.

Chung, F. R. (1997). *Spectral graph theory.* No. 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*, 277–296.

Gentile, C., & Warmuth, M. K. (1998). Linear hinge loss and average margin. *NIPS 1998, Proc.* (pp. 225–231). MIT Press.

Herbster, M. (2001). Learning additive models online with fast evaluating kernels. *COLT 2001, Proc.* (pp. 444–460). Springer.

Herbster, M., Pontil, M., & Wainer, L. (2005). *Online learning over graphs* (Working Paper). University College London.

Kondor, R., & Lafferty, J. (2002). Diffusion kernels on graphs and other discrete input spaces. *ICML 2002, Proc.* (pp. 315–322). Morgan Kaufmann.

Shalev-Shwartz, S., Crammer, K., Dekel, O., & Singer, Y. (2004). Online passive-aggressive algorithms. *NIPS 16.* MIT Press.

Smola, A., & Kondor, R. (2003). Kernels and regularization on graphs. *COLT 2003, Proc.* (pp. 144–158). Springer.

Tong, S., & Koller, D. (2000). Support vector machine active learning with applications to text classification. *ICML 2000, Proc.* (pp. 999–1006). Morgan Kaufmann.

Wahba, G. (1990). *Splines models for observational data*, vol. 59 of *Regional conference series in Applied Mathematics.* SIAM.

Warmuth, M. K., Liao, J., Rätsch, G., Mathieson, M., Putta, S., & Lemmen, C. (2003). Active learning with support vector machines in the drug discovery process. *Journal of Chemical Information and Computer Sciences*, *43*, 667–673.

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003a). Semi-supervised learning using gaussian fields and harmonic functions. *ICML 2003, Proc.* (pp. 912–919). AAAI Press.

Zhu, X., Lafferty, J., & Ghahramani, Z. (2003b). Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. *Proc. of the ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in ML and Data Mining* (pp. 58–65).