Online Similarity Prediction of Networked Data from Known and Unknown Graphs

Claudio Gentile   CLAUDIO.GENTILE@UNINSUBRIA.IT
DiSTA, Università dell'Insubria, Italy

Mark Herbster   M.HERBSTER@CS.UCL.AC.UK
Stephen Pasteris   S.PASTERIS@CS.UCL.AC.UK
Department of Computer Science, University College London, UK

Abstract

We consider online similarity prediction problems over networked data. We begin by relating this task to the more standard class prediction problem, showing that, given an arbitrary algorithm for class prediction, we can construct an algorithm for similarity prediction with "nearly" the same mistake bound, and vice versa. After noticing that this general construction is computationally infeasible, we target our study to *feasible* similarity prediction algorithms on networked data. We initially assume that the network structure is *known* to the learner. Here we observe that Matrix Winnow (Warmuth, 2007) has a near-optimal mistake guarantee, at the price of cubic prediction time per round. This motivates our effort for an efficient implementation of a Perceptron algorithm with a weaker mistake guarantee but with only poly-logarithmic prediction time. Our focus then turns to the challenging case of networks whose structure is initially *unknown* to the learner. In this novel setting, where the network structure is only incrementally revealed, we obtain a mistake-bounded algorithm with a quadratic prediction time per round.

Keywords: graph transduction, similarity prediction, online learning

1. Introduction

The study of networked data has spurred a large amount of research efforts. Applications like spam detection, product recommendation, link analysis, community detection, are by now well-known tasks in Social Network analysis and E-Commerce. In all these tasks, networked data are typically viewed as graphs, where vertices carry some kind of relevant information (e.g., user features in a social network), and connecting edges reflect a form of semantic similarity between the data associated with the incident vertices. Such a similarity ranges from friendship among people in a social network to common user's reactions to online ads in a recommender system, from functional relationships among proteins in a protein-protein interaction network to connectivity patterns in a communication network. Coarsely speaking, similarity prediction aims at inferring the existence of new pairwise relationships based on known ones. These pairwise constraints, which specify whether two objects belong to the same class or not, may arise directly from domain knowledge or be available with little human effort.

There is a wide range of possible means of capturing the structure of a graph in this learning context: through combinatorial and classical graph-theoretical methods (e.g., Gross

and Yellen (2003)); through spectral approachs (e.g. Belkin et al. (2004); Hein et al. (2007)), using convex duality and resistive geometry (e.g., Herbster and Lever (2009)), and even algebraic methods (e.g., Kondor et al. (2009)). In many of these approaches, the underlying assumption is that the graph structure is largely known in advance (a kind of "transductive" learning setting), and serves as a way to bias the inference process, so as to implement the principle that "connected vertices tend to be similar." Yet, this setting is oftentimes unrealistic and/or infeasible. For instance, a large online social network with millions of vertices and tens of millions of edges hardly lends itself to be processed as a whole via a Laplacian-regularized optimization approach or, even if it does (thanks to the computationally powerful tools currently available), it need not be known ahead of time. As a striking example, if we are representing a security agency, and at each point in time we receive a "trace" of communicating individuals, we still might want to predict whether a given pair in the trace belong to the same "gang"/community, even if the actual network of relationships is unknown to us. So, in this case, we are incrementally learning similarity patterns among individuals while, at the same time, exploring the network. Another important scenario of an unknown network structure is when the network itself *grows* with time, hence the prediction algorithms are expected to somehow adapt to its temporal evolution.

Our results. We study online similarity prediction over graphs in two models. One in which the graph is *known*[1] a priori to the learner, and one in which it is *unknown.* In both settings there is an *undisclosed* labeling of a graph so that each vertex is the member of one of K classes. Two vertices are similar if they are in the same class and dissimilar otherwise. The learner receives an online sequence of vertex pairs and similarity feedback. On the receipt of a pair the learner then predicts if the pair is similar. The true pair label, SIMILAR or DISSIMILAR, is then received and the goal of the learner is to minimize mistaken predictions. Our aim in both settings is then to bound the number of prediction mistakes over an arbitrary (and adversarially generated) sequence of pairs.

In the model where the graph is known, we first characterize the obtainable bounds for similarity prediction. This is accomplished by showing that our online similarity prediction problem *reduces* (Theorem 1) to online class prediction on graphs (e.g., Herbster (2008); Herbster and Lever (2009); Herbster and Pontil (2007); Herbster et al. (2005, 2009, 2008); Cesa-Bianchi et al. (2009, 2010a,b); Vitale et al. (2011), and references therein), and vice versa. Thus lower and upper bounds for class prediction transfer to similarity prediction. We then review prior upper and lower bounds for class prediction on graphs in Section 3.1, and apply them to similarity prediction in Proposition 2. Hence, Theorem 1 and Proposition 2 give us a view of *what is obtainable* for similarity prediction. However, the above reductions do not provide *efficient* computational constructions (which is a major concern here). The problem then becomes to obtain bounds for algorithms which are computationally efficient. As an interlude, Theorem 3 shows that a standard technique of using a particular Bayes classifier which was computationally efficient for class prediction on graphs is no longer likely to be so for similarity prediction. Now motivated to obtain computationally efficient algorithms, we resort to matrix learning algorithms. We show that a suitable adaptation of the Matrix Winnow algorithm (Warmuth, 2007) readily provides an almost optimal mistake

---

1. The reader should keep in mind that while the data at hand may not be natively graphical, it might still be convenient in practice to artificially generate a graph for similarity prediction, since the graph may encode side information that is otherwise unexploitable.

bound. This adaptation amounts to sparsifying the underlying graph G via a random spanning tree, whose diameter is then reduced by a known rebalancing technique (Herbster et al., 2009; Cesa-Bianchi et al., 2010a). Unfortunately, with its computational burden (cubic time per round), the resulting algorithm does not provide a satisfactory answer to deployment on large networks. Therefore, we develop an analogous adaptation of a Matrix Perceptron algorithm that delivers a more attractive answer (thanks to its poly-logarithmic time per round), though with an inferior online prediction performance guarantee.

The unknown model is identical to the known one, except that the learner does not initially receive the underlying graph G. Rather, G is incrementally revealed, as now when the learner receives a pair it also receives as side information an adversarially generated path within G connecting the vertices of the pair. Here, we observe that the machinery we used for the known graph case is inapplicable. In this novel setting, we design and analyze (Theorem 7) an algorithm which may be interpreted as a matrix version of an adaptive p-norm Perceptron (Grove et al., 2001; Gentile, 2003) with the relatively efficient quadratic running time per round.

Related work. This paper lies at the intersection between online learning on graphs and matrix/metric learning. Both fields include a substantial amount of work, so we can hardly do it justice here. Below we outline some of the main contributions in matrix/metric learning, with a special emphasis on those we believe are most related to this paper. Relevant papers in online class prediction on graphs will be recalled in Section 3.

Similarity prediction on graphs can be seen as a special case of matrix learning. Relevant works on this subject include Tsuda et al. (2005); Warmuth (2007); Cavallanti et al. (2010); Kakade et al. (2012) – see also Hazan et al. (2012) for recent usage in the context of online cut prediction. In all these papers, special care is put into designing appropriate regularization terms driving the online optimization problem, the focus typically being on spectral sparseness. When operating on graph structures with Laplacian-based regularization, these algorithms achieve mistake bounds depending on functions of the cut-size of the labeled graph – see Section 4. Yet, in the absence of further efforts, their scaling properties make them inappropriate to practical usage in large networks. *Metric* learning is also relevant to this paper. Metric learning is a special case of matrix learning where the matrix is positive semi-definite. Relevant references include Xing et al. (2002); Shalev-Shwartz et al. (2004); Maurer (2008); Zhang and Yan (2007); Cao et al. (2012). Some of these papers also contain generalization bound arguments. Yet, no specific concerns are cast on networked data frameworks. Related to our bidirectional reduction from class prediction to similarity prediction is the thread of papers on kernels on pairs (e.g., Basilico and Hofmann (2004); Maurer (2008); Li et al. (2008); Brunner et al. (2012)), where kernels over pairs of objects are constructed as a way to measure the "distance" between the two referenced pairs. The idea is then to combine with any standard kernel algorithm. The so-called matrix completion task (specifically, the recent reference Koltchinskii and Rangel (2012)) is also related to our work. In that paper, the authors introduce a matrix recovery method working in noisy environments, which incorporates both a low-rank and a Laplacian-regularization term. The problem of recovery of low-rank matrices has extensively been studied in the recent statistical literature (e.g., Candes and Recht (2009); Candes and Tao (2010); Gross (2011); Rohde and Tsybakov (2011); Negahban and Wainwright (2010); Koltchinskii et al. (2011), and references therein), the main concern being bounding the recovery error rate,

but disregarding the computational aspects of the selected estimators. Moreover, the way they typically measure error rate is not easily comparable to online mistake bounds. Finally, the literature on semisupervised clustering/clustering with side information (Ben-Dor et al. (1999); Demiriz et al. (1999) – see also Rangapuram and Hein (2012) for a recent reference on spectral approaches to clustering) is related to this paper, since the similarity feedback can be interpreted as a must-link/cannot-link feedback. Nonetheless, their formal statements are fairly different from ours.

To summarize, whereas we are motivationally close to Koltchinskii and Rangel (2012), from a technical viewpoint, we are perhaps closer to Shalev-Shwartz et al. (2004); Tsuda et al. (2005); Warmuth (2007); Cavallanti et al. (2010); Hazan et al. (2012); Kakade et al. (2012), as well as to the literature on online learning on graphs.

Before delving into the graph-based similarity problem, we start off by investigating the problem of similarity prediction in abstract terms, showing that similarity prediction reduces to classification, and vice versa. This will pave the way for all later results.

2. Online class and similarity prediction

In this section we examine the correspondence in predictive performance (mistake bounds) between the classification and similarity prediction frameworks.

Preliminaries. The set of all finite sequences from a set $\mathcal{X}$ is denoted $\mathcal{X}^*$. We use the Iverson bracket notation [PREDICATE] $= 1$ if the predicate is true and [PREDICATE] $= 0$ if false. In K-*class prediction* in the online mistake bound model, an *example sequence* $(x_1, y_1), \ldots, (x_T, y_T) \in (\mathcal{X} \times \mathcal{Y})^*$ is revealed incrementally, where $\mathcal{X}$ is a set of *patterns* and $\mathcal{Y} := \{1, \ldots, K\}$ is the set of K class *labels*. The goal on the t-th trial is to predict the class y_t given the previous $t-1$ pattern/label pairs and x_t. The overall aim of an algorithm is to minimize the number of its mistaken predictions. In *similarity prediction*, examples are pairs of patterns with "similarity" labels, i.e., $((x', x''), y) \in \mathcal{X}^2 \times \mathcal{Y}_s$ with $\mathcal{Y}_s = \{0, 1\}$. We interpret $y \in \mathcal{Y}_s$ as SIMILAR if $y = 0$ and DISSIMILAR if $y = 1$; we also introduce the convenient function $\text{sim}(y', y'') := 1 - [y' = y'']$ which maps a pair of class labels $y', y'' \in \mathcal{Y}$ to a similarity label. A *concept* is a function $f : \mathcal{X} \to \mathcal{Y}$ that maps patterns to labels. An example sequence S is *consistent* with a concept f for classification if $(x, y) \in S$ implies $y = f(x)$ and for similarity if $((x', x''), y) \in S$ implies $y = \text{sim}(f(x'), f(x''))$. We use $M_A(S)$ to denote the number of prediction mistakes of the *online algorithm A* on example sequence S. Given an algorithm A, we define the *mistake bound* with respect to a concept f as $\mathbb{B}_A(f) := \max_S M_A(S)$, the maximum being over all sequences S consistent with f.

Theorem 1 *Given an online* classification *algorithm* A_c *one may construct a* similarity *algorithm* A_s *such that if* S *is any* similarity *sequence consistent with any concept* f *then*

$$M_{A_s}(S) \le 5 \, \mathbb{B}_{A_c}(f) \log_2 K \,, \tag{1}$$

and given an online similarity *algorithm* A_s *one may construct a* classification *algorithm* A_c *such that if* S *is any* classification *sequence consistent with any concept* f *then*

$$M_{A_c}(S) \le \mathbb{B}_{A_s}(f) + K \,. \tag{2}$$

4

The direct implementation of the similarity algorithm A_s from the classification algorithm A_c is infeasible, as its running time is exponential in the mistake bound. In Appendix A.1 we prove a more general result (see Lemma 10) than in equation (1) which applies also to noisy sequences and to "order-dependent" bounds, as in the shifting-expert bounds in Herbster and Warmuth (1998). We also argue (Appendix A.1.1) that the "$\log K$" term in (1) is necessary. Observe that equation (2) implies a lower bound for similarity prediction if we have a lower bound for the corresponding class prediction problem with only a weakening by an additive "$-K$" term.

3. Class and similarity prediction on graphs

We now introduce notation specific to the graph setting. Let then $G = (V, E)$ be an undirected and connected graph with $n = |V|$ vertices, $V = \{1, \ldots, n\}$, and $m = |E|$ edges. The assignment of K class labels to the vertices of a graph is denoted by a vector $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$, where $y_i \in \{1, \ldots, K\}$ denotes the label of the i-th vertex among the K possible labels. The vertex-labeled graph will often be denoted by the pairing $(G, \boldsymbol{y})$. Associated with each pair $(i, j) \in V^2$ of (not necessarily adjacent) vertices is a similarity label $y_{i,j} \in \{0, 1\}$, where $y_{i,j} = 1$ if and only if $y_i \neq y_j$. As is typical of graph-based prediction problems (e.g., Herbster (2008); Herbster and Lever (2009); Herbster and Pontil (2007); Herbster et al. (2005, 2009, 2008); Cesa-Bianchi et al. (2009, 2010a,b); Vitale et al. (2011), and references therein), the graph structure plays the role of an inductive bias, where adjacent vertices tend to belong to the same class. The set of *cut-edges* in $(G, \boldsymbol{y})$ is denoted as $\Phi^G(\boldsymbol{y}) := \{(i, j) \in E : y_i \neq y_j\}$ (when nonambiguous, we abbreviate it to Φ^G), and the associated cut-size as $|\Phi^G(\boldsymbol{y})|$. The set of cut-edges with respect to class label k is denoted as $\Phi_k^G(\boldsymbol{y}) := \{(i, j) \in E : k \in \{y_i, y_j\}, y_i \neq y_j\}$ (when nonambiguous, we abbreviate it to Φ_k^G). Notice that $\sum_{k=1}^K |\Phi_k^G(\boldsymbol{y})| = 2|\Phi^G(\boldsymbol{y})|$. Let A be the $n \times n$ matrix such that $A_{ij} := A_{ji} = 1$ if $(i, j) \in E(G)$ and $A_{ij} := 0$ otherwise. Let D be the $n \times n$ diagonal matrix such that D_{ii} is the degree (in G) of vertex i. The Laplacian, L, of G is defined as $D - A$ and note that if $\boldsymbol{y} \in \{0, 1\}^n$ then we have that $|\Phi^G(\boldsymbol{y})| = \boldsymbol{y}^\top L \boldsymbol{y}$. We denote by L^+ the pseudoinverse of L. Given a symmetric positive semidefinite matrix X, we denote by $\sqrt{X}$ a symmetric positive semidefinite matrix in which $X = \sqrt{X}\sqrt{X}$.

If G is identified with a resistive network such that each edge is a unit resistor, then the *effective resistance* $R_{i,j}^G$ between a pair of vertices $(i, j) \in V^2$ can be defined as $R_{i,j}^G = (\boldsymbol{e}_i - \boldsymbol{e}_j)^\top L^+ (\boldsymbol{e}_i - \boldsymbol{e}_j)$, where $\boldsymbol{e}_i$ is the i-th vector in the canonical basis of $\mathbb{R}^n$. When $(i, j) \in E$ then $R_{i,j}^G$ also equals the probability that a spanning tree of G drawn uniformly at random (from the set of all spanning trees of G) includes (i, j) as one of its $n - 1$ edges (e.g., Lyons and Peres (2012)). The resistance diameter of G is $\max_{(i,j) \in V^2} R_{i,j}^G$. It is known that the effective resistance defines a metric over the vertices of G. Moreover, when G is a tree, then $R_{i,j}^G$ corresponds to the number of edges in the (unique) path from i to j. Hence, in this case, the resistance diameter of G coincides with its (geodesic) diameter.

3.1. Class prediction on graphs

Roughly speaking, algorithms and bounds for sequential class prediction on graphs split between two types: Those which approximate the original graph with a tree or those that

maintain the original graph. By approximating the graph with a tree, extremely efficient algorithms are obtained with strong optimality guarantees. By exploiting the full graph, algorithms are obtained which take advantage of the connectivity to achieve sharp bounds when the graph contains, e.g., dense clusters. Relevant literature on this subject includes Herbster et al. (2005); Herbster and Pontil (2007); Herbster (2008); Herbster et al. (2008, 2009); Cesa-Bianchi et al. (2009); Herbster and Lever (2009); Cesa-Bianchi et al. (2010a). Known representatives of the first kind are upper bounds of the form $\mathcal{O}(|\Phi^T|(1 + \log \frac{n}{|\Phi^T|}))$ (Herbster et al., 2009) or of the form $\mathcal{O}(|\Phi^T| \log D_T)$ (Cesa-Bianchi et al., 2009), where T is some spanning tree of G, and D_T is the (geodesic) diameter of T. In particular, if T is drawn uniformly at random, the above turn to bounds on the *expected* number of mistakes of the form $\mathcal{O}(\mathbb{E}[|\Phi^T|] \log n)$, where $\mathbb{E}[|\Phi^T|]$ is the resistance-weighted cut-size of G, $\mathbb{E}[|\Phi^T|] = \sum_{(i,j) \in \Phi^G(\boldsymbol{y})} R_{i,j}^G$, which can be far smaller than $|\Phi^G|$ when G is well connected. Representatives of the second kind are bounds of the form $\mathcal{O}(\rho + |\Phi^G| R_\rho)$ (Herbster, 2008; Herbster and Lever, 2009), where ρ is the number of balls in a cover of the vertices of G such that R_ρ is the maximum over the resistance diameters of the balls in the cover. Since resistance diameter lower bounds geodesic diameter, this alternative approach leverages a different connectivity structure of the graph than the resistance-weighted cut-size.

In the references above the bounds and algorithms are for the two-class prediction case. In Appendix A.2 we give a reduction that raises a variety of cut-based algorithms and bounds from the two-class to the K-class case. Specifically, a two-class mistake bound of the form $M \le \max(c|\Phi^G(\boldsymbol{y})|, 1) \ \forall \boldsymbol{y} \in \{1, 2\}^n$, for some $c \ge 1$ becomes a K-class bound of the form $M \le \max(2c|\Phi^G(\boldsymbol{y})|, 1) \ \forall \boldsymbol{y} \in \{1, \ldots, K\}^n$, where K is not known in advance to the algorithm. Thus, bounds of the form $\mathcal{O}(\mathbb{E}[|\Phi^T|] \log n)$ also hold in the multiclass setting.

On the lower bound side, Cesa-Bianchi et al. (2009) contains an argument showing (for the $K = 2$ class case) that for any $\phi \ge 0$ a labeling $\boldsymbol{y}$ exists such that any algorithm will make at least $\phi/2$ mistakes while $\mathbb{E}[|\Phi^T|] < \phi$. In short, $\Omega(\mathbb{E}[|\Phi^T|])$ is also a lower bound on the number of mistakes in the class prediction problem on graphs. When combined with Theorem 1 in Section 2, the above results immediately yield upper and lower bounds for the similarity prediction problem over graphs.

Proposition 2 *Let $(G, \boldsymbol{y})$ be a labeled graph, and T be a random spanning tree of G. Then an algorithm exists for the similarity prediction problem on G whose expected number of mistakes $\mathbb{E}[M]$ satisfies $\mathbb{E}[M] = \mathcal{O}(\mathbb{E}[|\Phi^T(\boldsymbol{y})|] \log K \log n)$. Moreover, for any $\phi \ge 0$ a K-class labeling $\boldsymbol{y}$ exists such that any similarity prediction algorithm on G will make at least $\phi/2 - K$ mistakes while $\mathbb{E}[|\Phi^T|] < \phi$.*

The upper bound above refers to a computationally inefficient algorithm and, clearly enough, more direct version space arguments would lead to similar results. Section 4 contains a more efficient approach to similarity prediction over graphs.

To close this section, we observe that the upper bounds on class predictions of the form $\mathcal{O}(\mathbb{E}[|\Phi^T|] \log n)$ taken from Herbster et al. (2009); Cesa-Bianchi et al. (2010a) are essentially relying on linearizing the graph G into a path graph, and then predicting optimally on it via an efficient Bayes classifier (aka Halving Algorithm, e.g., Littlestone (1988)). One might wonder whether a similar approach would directly apply to the similarity prediction problem. We now show that exact computation of the probabilities of the Bayes classifier for a path graph is #P-complete under similarity feedback.

6

The ISING distribution over graph labelings ($\boldsymbol{y} \in \{1, 2\}^n$) is defined as $p(\boldsymbol{y}) \propto 2^{-\beta|\phi^G(\boldsymbol{y})|}$. Given a set of vertices and associated labels, the marginal distribution at each vertex can be computed in linear time when the graph is a path (Pearl (1982)). In Herbster et al. (2009) this simple fact was exploited to give an efficient class prediction algorithm by a particular linearization of a graph to a path graph. The equivalent problem in similarity prediction requires us to compute marginals given a set of pairwise constraints. The following theorem shows that computing the partition function (and hence the relevant marginals) of the ISING distribution on a path with pairwise label constraints is #P-complete.

Theorem 3 *Computing the partition function of the (ferromagnetic) ISING model on a path graph with pairwise constraints is #P-complete, where an INSTANCE is an n-vertex path graph P, a set of pairs $\mathcal{C} \subset \{1, \ldots, n\}^2$, and a natural number, β, presented in unary, and the desired OUTPUT is the value of the partition function, $Z_P(\mathcal{C}, \beta) := \sum_{\boldsymbol{y} \in \{1,2\}^n : \{(y_i = y_j)\}_{(i,j) \in \mathcal{C}}} 2^{-\beta|\phi^P(\boldsymbol{y})|}$.*

Thus computing the *exact* marginal probabilities on even a path graph will be infeasible (given the hardness of #P). As an alternative, in the following section we discuss the application of the Matrix Perceptron and Matrix Winnow algorithms to similarity prediction.

3.2. Similarity prediction on graphs

In Algorithm 1 we give a simple application of the Matrix Winnow (superscript "w") and Perceptron (superscript "p") algorithms to similarity prediction on graphs. The key aspect of the construction (common to many methods in metric learning) is the creation of rank-one matrices which correspond to similarity "instances" (see (3)), with the idea of turning the similar/dissimilar feedback into a (matrix) linear-threshold constraint. Specifically, the encoding for a vertex pair (i_t, j_t) as matrix X_t may be motivated by observing that the inner product of the target matrix with the instance matrix X_t (see (9) in the appendix) is 0 if i_t and j_t share the same class label, and is 2 otherwise. In addition, we need to regularize our algorithms so as to let them bet on a small cutsize of $(G, \boldsymbol{y})$. This results in Matrix Perceptron and Matrix Winnow algorithms run with a kernel defined by the pseudoinverse Laplacian matrix of the graph, both algorithms being engaged in learning low-rank matrices. We then use the standard analysis of the Perceptron (Novikoff, 1962) and Matrix Winnow (Warmuth, 2007) algorithms with appropriate thresholds to obtain Proposition 4. The mistake bounds therein follow from margin analyses of linear-threshold algorithms. A key observation is that the squared Frobenius norm of the (un-normalized) instance matrices is bounded by the squared resistance diameter of the graph, and the squared Frobenius norm of the (un-normalized) "comparator" matrix is bounded by the cut-size squared $|\Phi^G|^2$. Here, small cut-size can be interpreted as the inductive bias of our algorithms, the idea is that the smaller the cut-size then the better the expected performance of our algorithms.

Proposition 4 *Let $(G, \boldsymbol{y})$ be a labeled graph. If we run the Matrix Winnow and Perceptron algorithms with G as the input graph, we have the following mistake bounds:*

$$M^{\mathrm{W}} = \mathcal{O}\left(|\Phi^G| \max_{(i,j) \in V^2} R_{i,j}^G \log n\right) \quad and \quad M^{\mathrm{P}} = \mathcal{O}\left(|\Phi^G|^2 \max_{(i,j) \in V^2} (R_{i,j}^G)^2\right).$$

Algorithm 1: Perceptron and Matrix Winnow algorithms on a graph

---

Input: Graph $G = (V, E)$, $|V| = n$, with Laplacian L, and $R := \max_{(i,j) \in V^2} R_{i,j}^G$;

Parameters: Perceptron threshold $\widehat{\theta}^{\mathrm{p}} = R^2$; Winnow threshold $\widehat{\theta}^{\mathrm{w}} = \frac{\eta}{e^\eta - e^{-\eta}} \frac{1}{R |\Phi^G|}$,
Winnow learning rate $\eta = 1.28$;

Initialization: $W_0^{\mathrm{p}} = \mathbf{0} \in \mathbb{R}^{n \times n}$; $W_0^{\mathrm{w}} = \frac{1}{n} I \in \mathbb{R}^{n \times n}$;

For $t = 1, 2, \ldots, T$:

- Get pair of vertices $(i_t, j_t) \in V^2$, and construct similarity instances,

$$X_t^{\mathrm{p}} = \sqrt{L^+}(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^\top \sqrt{L^+}; \quad X_t^{\mathrm{w}} = \frac{\sqrt{L^+}(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^\top \sqrt{L^+}}{(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^\top L^+(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})}; \quad (3)$$

- Predict: $\hat{y}_t^{\mathrm{p}} = [\mathrm{TR}((W_{t-1}^{\mathrm{p}})^\top X_t^{\mathrm{p}}) > \widehat{\theta}^{\mathrm{p}}]$; $\hat{y}_t^{\mathrm{w}} = [\mathrm{TR}((W_{t-1}^{\mathrm{w}})^\top X_t^{\mathrm{w}}) > \widehat{\theta}^{\mathrm{w}}]$;

- Observe $y_t \in \{0, 1\}$ and, if mistake ($y_t \neq \hat{y}_t$), update

$$W_t^{\mathrm{p}} \leftarrow W_{t-1}^{\mathrm{p}} + (y_t - \hat{y}_t^{\mathrm{p}}) X_t^{\mathrm{p}}; \qquad \log W_t^{\mathrm{w}} \leftarrow \log W_{t-1}^{\mathrm{w}} + \eta (y_t - \hat{y}_t^{\mathrm{w}}) X_t^{\mathrm{w}} .$$

A severe drawback of both these algorithms is that on a generic graph, initilization requires computing a pseudo-inverse (typically cubic time), and furthermore the update of Matrix Winnow requires a cubic-time computation of an eigendecomposition (to compute matrix exponentials) on each mistaken trial.[2] In the following section, we focus on a construction based on a graph approximation for which we develop an efficient implementation of the Perceptron algorithm which will require only poly-logarithmic time per round.

4. Efficient similarity prediction on graphs

Relying on the notation of Section 3, we turn to efficient similarity prediction on graphs. We present adaptations of Matrix Winnow and Matrix Perceptron to the case when the original graph G is sparsified through a linearized and rebalanced random spanning tree of G. This sparsification technique, called a Binary Support Tree (BST) in (Herbster et al., 2009), brings the twofold advantage of yielding improved mistake bounds and faster prediction algorithms. More specifically, the use of a BST replaces the (perhaps very large) resistance diameter term $\max_{(i,j) \in V^2} R_{i,j}^G$ in the mistake bounds of Proposition 4 by a logarithmic term, the other term in the mistake bound becoming (when dealing with the expected number of mistakes) only a logarithmic factor larger than the (often far smaller) sum of the resistance-weighted cut-sizes in a spanning tree. Moreover, when combined with the Perceptron algorithm, a BST allows us to develop a very fast implementation whose running time per round is poly-logarithmic in n, rather than cubic, as in Matrix Winnow-like algorithms.

Recall that a uniformly random spanning tree of an unweighted graph can be sampled in expected time $\mathcal{O}(n \ln n)$ for "most" graphs Broder (1989). Using the nice algorithm of

---

2. Additionally, there is a tuning issue related to Matrix Winnow, since the threshold $\widehat{\theta}^{\mathrm{w}}$ depends on the (unknown) cut-size.
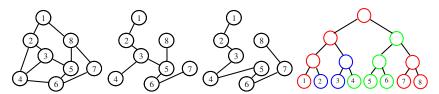
Figure 1: **From left to right:** The graph G; a random spanning tree T of G (note that the vertices are numbered by the order of a depth-first visit of T, starting from root vertex 1; the path graph P which follows the order of the depth-first visit of T; the BST built on top of P. Notice how the class labels of the 8 vertices in V (corresponding to the three colors) are propagated upwards.

Wilson (1996), the expected time reduces to $\mathcal{O}(n)$ —see also the work of Alon et al. (2011). However, all known techniques take expected time $\Theta(n^3)$ in certain pathological cases.

In a nutshell, a BST B of G is a full balanced binary tree whose leaves correspond to the vertices in[3] V. In order to construct B from G, we first extract a random spanning tree T of G, then we visit T through a depth-first visit, order its vertices according to this visit eliminating duplicates (thereby obtaining a path graph P), and finally we build B on top of P. Since B has $2n-1$ vertices, we extend the class labels from leaves to internal vertices by letting, for each internal vertex i of B, y_i be equal to the class label of i's left child. Figure 1 illustrates the process. A simple adaptation of Herbster et al. (2009) (Section 6 therein) shows that for any class $k = 1, \ldots, K$ we have $|\Phi_k^B| \le 2\,|\Phi_k^T|\,\log_2 n$. With the above handy, we can prove the following bounds (see Appendix A.3 for further details).

Theorem 5 *Let $(G, \boldsymbol{y})$ be a labeled graph, T be a random spanning tree of G and B be the corresponding BST.*

1. *If we run Matrix Winnow with B as the input graph (see Algorithm 1) then the expected number of mistakes $\mathbb{E}[M]$ on G satisfies $\mathbb{E}[M] = \mathcal{O}\left(\varphi\,\log^3 n\right)$,*

2. *and if we run the Matrix Perceptron algorithm with B as the input graph then $\mathbb{E}[M] = \mathcal{O}\left(\varphi^2\,\log^4 n\right)$,*

where we denote the resistance-weighted cut-size as $\varphi = \mathbb{E}[|\Phi^T|] = \sum_{(i,j)\in\Phi^G} R_{i,j}^G$.

The bound for Matrix Winnow is optimal up to a $\log^3 n$ factor — compare to the lower bound in Proposition 2. However, this tight bound is obtained at the cost of having an algorithm which is $\mathcal{O}(n^3)$ per round, even when run on a tree. This is because matrix exponentials require storing and updating a full SVD of the algorithm's weight matrix at each round, thereby making this algorithm highly impractical when G is large. On the other hand, the Perceptron bound is significantly suboptimal (due to its dependence on the squared resistance-weighted cut-size), but it has the invaluable advantage of lending itself to a very efficient implementation: Whereas a naive implementation would lead to an $\mathcal{O}(n^2)$ running time per round, we now show that a more involved implementation exists which takes only $\mathcal{O}(\log^2 n)$, yielding an exponential improvement in the per-round running time.

---

3. We assume w.l.o.g. that $n = |V|$ is a power of 2. Otherwise, we may add dummy "leaves".
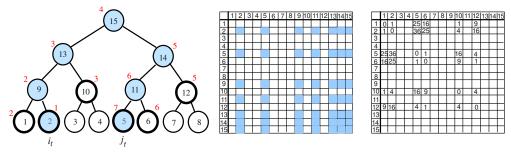
Figure 2: Matrix Perceptron algorithm at time t with $i_t = 2$ and $j_t = 5$. **Left:** The BST. Light blue vertices are those in $\mathcal{P}_t$. Thick-bordered vertices are those in $\mathcal{S}_t$. The f_t tags are the red numbers near the involved vertices (i.e., those in $\mathcal{P}_t \cup \mathcal{S}_t$). **Middle:** The matrix F. In light blue are the entries of F that are summed over in (4). **Right:** The matrix F, where numbers are the values $(f_t(\ell) - f_t(\ell'))^2$ that are added to $(y_{i_t,j_t} = 1, \hat{y}_{i_t,j_t} = 0)$ or subtracted from $(y_{i_t,j_t} = 0, \hat{y}_{i_t,j_t} = 1)$ the respective components of F during the update step (5).

4.1. Implementing Matrix Perceptron on BST

The algorithm operates on the BST B by maintaining a $(2n - 1) \times (2n - 1)$ symmetric matrix F with integer entries initially set to zero. At time t, when receiving the pair of leaves (i_t, j_t), the algorithm constructs $\mathcal{P}_t$, the (unique) path in B connecting i_t to j_t. Then the prediction $\hat{y}_{i_t,j_t} \in \{0, 1\}$ is computed as

$$\hat{y}_{i_t,j_t} = \begin{cases} 1 & \text{if } \sum_{\ell,\ell' \in \mathcal{P}_t} F_{\ell,\ell'} \geq 4 \log^2 n, \\ 0 & \text{otherwise .} \end{cases} \quad (4)$$

Upon receiving label y_{i_t,j_t}, the algorithm updates F as follows. First of all, the algorithm is mistake driven, so an update takes place only if $y_{i_t,j_t} \neq \hat{y}_{i_t,j_t}$. Let $\mathcal{N}_t$ be the set of neighbors of the vertices in $\mathcal{P}_t$, and define $\mathcal{S}_t := \mathcal{N}_t \setminus (\mathcal{P}_t \setminus \{i_t, j_t\})$. We recursively assign integer tags $f_t(\ell)$ to vertices $\ell \in \mathcal{N}_t$ as follows: 1. For all $\ell \in \mathcal{P}_t$, if ℓ is the s-th vertex in $\mathcal{P}_t$ then we set $f_t(\ell) = s$; 2. For all $\ell \in \mathcal{N}_t \setminus \mathcal{P}_t$, let n_ℓ be the (unique) neighbor of ℓ that is contained in $\mathcal{P}_t$. Then we set $f_t(\ell) = f_t(n_\ell)$. We then update F on each pair $(\ell, \ell') \in \mathcal{S}_t^2$ as

$$F_{\ell,\ell'} \leftarrow F_{\ell,\ell'} + (2y_{i_t,j_t} - 1)(f_t(\ell) - f_t(\ell'))^2 . \quad (5)$$

Figure 2 illustrates the process. The following theorem is the main technical result of this section. Its involved proof is given in Appendix A.3.

Theorem 6 *Let B be a BST of a labeled graph $(G, \boldsymbol{y})$ with $|V| = n$. Then the algorithm described by (4) and (5) is equivalent to Matrix Perceptron run with B as the input graph. Moreover, the algorithm takes $\mathcal{O}(\log^2 n)$ per trial, and there exists an adaptive representation of F with an initialisation time of only $\mathcal{O}(n)$ (rather than $\mathcal{O}(n^2)$).*
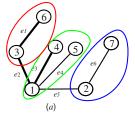
5. The Unknown Graph Case

We now consider the case when the graph $G = (V, E)$ is *unknown* to the learner beforehand. The graph structure is thus revealed incrementally as more and more pairs (i_t, j_t) get produced by the adversary. A reasonable online protocol that includes progressive graph

disclosure is the following. At the beginning of round $t = 1$ the learner knows nothing about G, but the number of vertices n — prior knowledge of n makes presentation easier, but could easily be removed from this setting. In a generic round t, the adversary presents to the learner both a pair $(i_t, j_t) \in V \times V$ and a path within G from i_t to j_t. The learner is then compelled to predict whether or not the two vertices are similar. Notice that, although the presented path may have cut-edges, there might be alternative paths in G connecting the two vertices with no cut-edges. The learner need not see them. The adversary then reveals the similarity label y_{i_t, j_t} in G, and the next round begins. In this setting, the adversary has complete knowledge of G, and can decide to produce paths and place the cut-edges in an adaptive fashion. Notice that, because of the incremental disclosure of G, no such constructions as Laplacian-based similarity instances and/or BST, as contained in Section 3.2 and Section 4, are immediately applicable.
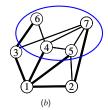
As a simple warm-up, consider the case when G is a tree and the K class label sets of vertices (henceforth called *clusters*) correspond to connected components of G. Figure 3 (a) gives an example. Since the graph is a tree, the number of cut-edges equals $K - 1$. We can associate with such a tree a linear-threshold function vector $\boldsymbol{u} = (u_1, \ldots, u_{n-1})^\top \in \{0, 1\}^{n-1}$, where u_i is 1 if and only if the i-th edge is a cut-edge. The ordering of edges within $\boldsymbol{u}$ can be determined ex-post by first disclosure times. For instance, if in round $t = 1$ the adversary produces pair $(6, 4)$ and path $6 \to 3 \to 1 \to 4$ (Figure 3 (a)), then edge $(6, 3)$ will be the first edge, $(3, 1)$ will be the second, and $(1, 4)$ will be the third. Then, if in round $t = 2$ the new pair is $(3, 5)$ and the associated path is $3 \to 1 \to 5$, the newly revealed edge $(1, 5)$ will be the fourth edge within $\boldsymbol{u}$. With this ordering in mind, the algorithm builds at time t the $(n - 1)$-dimensional vector $\boldsymbol{x}_t = (x_{1,t}, \ldots, x_{n-1,t})^\top \in \{0, 1\}^{n-1}$ corresponding to the path disclosed at time t, where $x_{i,t}$ is 1 if and only if the i-th edge belongs to the path. Now, it is clear that $y_{i_t, j_t} = 1$ if $\boldsymbol{u}^\top \boldsymbol{x}_t \geq 1$, and $y_{i_t, j_t} = 0$ if $\boldsymbol{u}^\top \boldsymbol{x}_t = 0$. Therefore, this turns out to be a sparse linear-threshold function learning problem, and a simple application of the standard Winnow algorithm (Littlestone, 1988) leads to an $\mathcal{O}(K \log n) = \mathcal{O}\left(|\Phi^G| \log n\right)$ mistake bound obtained by an efficient ($\mathcal{O}(n)$ time per round) algorithm, independent of the finer structural properties of G, such as its diameter.

One might wonder if an adaptation of the above procedure exists which applies to a general graph G by, say, extracting a spanning tree T out of G, and then applying the Winnow algorithm on T. Unfortunately, the answer is negative for at least two reasons. First, the above linear-threshold model heavily relies on the fact that clusters are connected, which need not be the case in our similarity problem. More critically, even if the clusters are connected in G, they need not be connected in T. Figure 3 (b)-(c) shows a typical example where Winnow applied to a spanning tree fails. Given this state of affairs, we are lead to consider a slightly different representation for pairs of vertices and paths. Yet, as before, this representation will suggest a linear separability condition, as well as the deployment of appropriate linear-threshold algorithms.

5.1. Algorithm and analysis

Algorithm 2 contains the pseudocode of our algorithm. When interpreted as operating on vectors, the algorithm is simply an r-norm perceptron algorithm (Grove et al., 2001; Gentile, 2003) with nonzero threshold, and norm $r = 2 \log(n - 1)^2 = 4 \log(n - 1)$, with
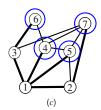
Figure 3: **(a)** A tree with 3 clusters corresponding to the 3 depicted connected components. Edges e_2 and e_5 are the cut-edges. Assuming edges are initially revealed in the order of their subscripts, the associated vector $\boldsymbol{u}$ is $\boldsymbol{u} = (0,1,0,0,1,0)^\top$. The algorithm receives at time $t = 1$ the pair $(6,4)$ along with path $6 \to 3 \to 1 \to 4$ (so that the 3 thick edges are revealed in the first round). The associated feature vector is $\boldsymbol{x}_1 = (1,1,1,0,0,0)^\top$. Vertices 6 and 4 are disconnected as $\boldsymbol{u}^\top \boldsymbol{x}_1 \geq 1$. **(b)-(c)** The connectivity structure induced by the thick-edged spanning tree on the blue cut in (b). Vertices 4, 5, 6, and 7 are all connected in G under the blue cut (b), but they are *all disconnected* in T (c).

$(n-1)^2$ as the length of the vectors maintained throughout, and s the dual to norm r. At time t, the algorithm observes pair (i_t, j_t) and path $p_{(i_t \to j_t)}$, builds the instance vector $\boldsymbol{x}_t \in \{-1, 0, 1\}^{n-1}$ and the long vector $\text{VEC}(X_t)$ out of the rank-one matrix $X_t = \boldsymbol{x}_t \boldsymbol{x}_t^\top$, where $\text{VEC}(\cdot)$ is the standard vectorization of a matrix that stacks its columns one underneath the other. In order to construct $\boldsymbol{x}_t$ from $p_{(i_t \to j_t)}$, the algorithm maintains a forest made up of the union of paths seen so far. If pair (i_t, j_t) is already connected by a path p in the current forest, then $\boldsymbol{x}_t$ is the instance vector associated with path p (as in the Winnow algorithm on a tree in the previous section, but taking edge orientations into account – see Figure 7 in Appendix A.4 for details). Otherwise, path $p_{(i_t \to j_t)}$ is added to the forest and $\boldsymbol{x}_t$ will be an instance vector associated with the new path $p_{(i_t \to j_t)}$. In adding the new path to the forest, we need to make sure that no circuits are generated. In particular, as soon as a revealed edge in a path causes two subtrees to join, the algorithm merges the two subtrees and processes all remaining edges in that path in a sequential manner so as to avoid generating circuits. The algorithm will end up using a spanning tree T of G for building its instance vectors $\boldsymbol{x}_t$. This spanning tree is determined on the fly by the adversarial choices of pairs and paths, so *it is not known to the algorithm* ahead of time.[4] But any later change to the spanning forest is designed so as to keep consistency with all previous vectors $\boldsymbol{x}_t$.

The decision threshold $(r-1)\|\boldsymbol{x}_t\|_r^4 = (r-1)\|\text{VEC}(X_t)\|_r^2$ follows from a standard analysis of the r-norm perceptron algorithm with nonzero threshold (easily adapted from Grove et al. (2001); Gentile (2003)), as well as the update rule. In short, since the graph is initially unknown, the algorithm is pretending to learn vectors rather than (Laplacian-regularized) matrices, and relies on a regularization that takes advantage of the sparsity of such vectors. The analysis of Theorem 7 below rests on ancillary (and classical) properties of matroids on graphs. These are recalled in Appendix A.4, before the proof of the theorem.

Theorem 7 *If Algorithm 2 is run on an arbitrary sequence of pairs $(i_1, j_1), (i_2, j_2), \dots$ with associated paths $p_{(i_1 \to j_1)}, p_{(i_2 \to j_2)}, \dots$ then we have the mistake bound $M = \mathcal{O}\left(|\Phi^G|^4 \log n\right)$.*

Remark 8 *As explained in the proof of Theorem 7, the separability condition (10) allows one to run any vector or matrix mirror descent linear-threshold algorithm. In particular,*

---

4. In fact, because the algorithm is deterministic, this spanning tree is fully determined by the adversary.

Algorithm 2: r-norm Perceptron for similarity prediction in unknown graphs.

Input: Number of vertices $n = |V|$, $V = \{1, \ldots, n\}$, $n \geq 3$;

Initialization : $\boldsymbol{w}_0 = \boldsymbol{0} \in \mathbb{R}^{(n-1)^2}$; $r = 4\log(n-1)$, $s = r/(r-1)$;

For $t = 1, 2, \ldots, T$:

- Get pair of vertices $(i_t, j_t) \in V^2$, and path $p_{(i_t \to j_t)}$. Construct instance vector $\boldsymbol{x}_t \in \{-1, 0, 1\}^{n-1}$ as explained in the main text;

- Build vector $\text{VEC}(X_t)$, where $X_t = \boldsymbol{x}_t \boldsymbol{x}_t^\top$, and predict $\hat{y}_t \in \{0, 1\}$ as

$$\hat{y}_t = \begin{cases} 1 & \text{if } \boldsymbol{w}_{t-1}^\top \text{VEC}(X_t) \geq (r-1) \, \|\boldsymbol{x}_t\|_r^4 \\ 0 & \text{otherwise,} \end{cases}$$

- Observe $y_t \in \{0, 1\}$ and, if mistake ($y_t \neq \hat{y}_t$), update

$$f(\boldsymbol{w}_t) \leftarrow f(\boldsymbol{w}_{t-1}) + (y_t - \hat{y}_t) \, \text{VEC}(X_t), \qquad f(\boldsymbol{w}) = \nabla \|\boldsymbol{w}\|_s^2 / 2 \ .$$

---

since matrix U therein is spectrally sparse (rank $K << n$), one could use unitarily invariant regularization methods, like (squared) trace norm-based online algorithms (e.g., Warmuth (2007); Cavallanti et al. (2010); Kakade et al. (2012)). For instance, Matrix Winnow (more generally, Matrix EG-like algorithms (Tsuda et al., 2005)) would get bounds which are linear in the cutsize but also (due to their unitary invariance) linear in $\|\boldsymbol{x}_t\|_2^2$. The latter can be as large as the diameter of T, which can easily be $\mathcal{O}(n)$ even if the diameter of G is much smaller. This makes these bounds significantly worse than Theorem 7 when the total cutsize $|\Phi^G|$ is small compared to n (which is our underlying assumption throughout). Group norm regularizers can also be used. Yet, because X_t has rank one, when $|\Phi^G|$ is small these regularizers do not lead to better bounds[5] than Theorem 7. Moreover, it is worth mentioning that, among the standard mirror descent linear-threshold algorithms operating on vectors $\text{VEC}(\cdot)$, our choice of the r-norm Perceptron is motivated by the fact this algorithm achieves a logarithmic bound in n with no prior knowledge of the actual cutsize $|\Phi^G|$ (or an upper bound thereof) – see Section 3.2, and the discussion in Gentile (2003) about tuning of parameters in r-norm Perceptron and Winnow/Weighted Majority-like algorithms.

As a final remark, our algorithm has an $\mathcal{O}(n^2)$ running time per round, trivially due to the update rule operating on $\mathcal{O}(n^2)$-long vectors. The construction of instance vector $\boldsymbol{x}_t$ out of path $p_{(i_t \to j_t)}$ can indeed be implemented faster than $\Theta(n^2)$ by maintaining well-known data structures for disjoint sets (e.g., (Cormen et al., 2001, Ch. 22)).

---

5. In fact, our r-norm algorithm with $\text{VEC}(\cdot)$ vectors is equivalent to a group norm regularization technique applied to the corresponding matrices where row and column norms are both equal to $s = r/(r-1)$.

References

N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, and M. Tuttle. Many random walks are faster than one. *Comb. Probab. Comput.*, 20(4):481–502, 2011.

J. Basilico and T. Hofmann. Unifying collaborative and content-based filtering. In *Proc of the 21st ICML*, ICML, 2004.

M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *COLT 2004*, 2004.

A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4), 1999.

A. Broder. Generating random spanning trees. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 442–447. IEEE Computer Society, 1989.

C. Brunner, A. Fischer, K. Luig, and T. Thies. Pairwise support vector machines and their application to large scale problems. *Journal of Machine Learning Research*, 13:2279–2292, 2012.

E. Candes and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

E. Candes and T. Tao. The power of convex relaxation: near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56:2053–2080, 2010.

Q. Cao, Z. Guo, and Y. Ying. Generalization bounds for metric and similarity learning. *CoRR*, abs/1207.5437, 2012.

G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Linear algorithms for online multitask classification. *Journal of Machine Learning Research*, 11:2901–2934, 2010.

N. Cesa-Bianchi, C. Gentile, and F. Vitale. Fast and optimal prediction on a labeled tree. In *Proceedings of the 22nd Annual Conference on Learning*. Omnipress, 2009.

N. Cesa-Bianchi, C Gentile, F. Vitale, and G. Zappella. Random spanning trees and the prediction of weighted graphs. In *Proceedings of the 27th International Conference on Machine Learning (27th ICML)*, pages 175–182, 2010a.

N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. Active learning on trees and graphs. In *Proceedings of the 23rd Conference on Learning Theory (23rd COLT)*, pages 320–332, 2010b.

T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001. ISBN 0070131511.

A. Demiriz, K. Bennett, and M.J. Embrechts. Semi-supervised clustering using genetic algorithms. In *In Artificial Neural Networks in Engineering (ANNIE-99)*, pages 809–814, 1999.

C. Gentile. The robustness of the p-norm algorithms. *Machine Learning*, 53:265–299, 2003.

D. Gross. Recovering low-rank matrices from few coefficients in any basis. *IEEE Transactions on Information Theory*, 57/3:1548–1566, 2011.

J. L. Gross and J. Yellen. *Handbook of graph theory*. CRC Press, 2003. ISBN 1584880902.

A. J. Grove, N. Littlestone, and D. Schuurmans. General convergence results for linear discriminant updates. *Machine Learning*, 43:173–210, 2001.

E. Hazan, S. Kale, and S. Shalev-Shwartz. Near-optimal algorithms for online matrix prediction. In *Proceedings of the 25th Annual Conference on Learning Theory (COLT'12)*, 2012.

M. Hein, J.Y. Audibert, and U. von Luxburg. Graph laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8:1325–1368, 2007.

M. Herbster. Exploiting cluster-structure to predict the labeling of a graph. In *Proceedings of the 19th International Conference on Algorithmic Learning Theory*, pages 54–69, 2008.

M. Herbster and G. Lever. Predicting the labelling of a graph via minimum p-seminorm interpolation. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT'09)*, 2009.

M. Herbster and M. Pontil. Prediction on a graph with a perceptron. In *Advances in Neural Information Processing Systems 19*, pages 577–584. MIT Press, Cambridge, MA, 2007.

M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32(2): 151–178, 1998. ISSN 0885-6125.

M. Herbster, M. Pontil, and L. Wainer. Online learning over graphs. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 305–312, New York, NY, USA, 2005. ACM.

M. Herbster, M. Pontil, and S. R. Galeano. Fast prediction on a tree. In *Proc. of the 22nd Annual Conference on Neural Information Processing Systems*, pages 657–664. MIT Press, 2008.

M. Herbster, G. Lever, and M. Pontil. Online prediction on large diameter graphs. In *Advances in Neural Information Processing Systems (NIPS 22)*, pages 649–656. MIT Press, 2009.

M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.

S. M. Kakade, S. Shalev-Shwartz, and A. Tewari. Regularization techniques for learning with matrices. *The Journal of Machine Learning Research*, pages 1865–1890, 2012.

V. Koltchinskii and P. Rangel. Low rank estimation of similarities on graphs. *CoRR*, 2012.

V. Koltchinskii, K. Lounici, and A. Tsybakov. Nuclear norm penalization and optimal rates for noisy matrix completion. *Annals of Statistics*, 39(5):2302–2329, 2011.

R. Kondor, N. Shervashidze, and K. M. Borgwardt. The graphlet spectrum. In *ICML 2009*, 2009.

Z. Li, J. Liu, and X. Tang. Pairwise constraint propagation by semidefinite programming for semi-supervised classification. In *Proc of the 25th ICML*, ICML, 2008.

N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, April 1988.

N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108 (2):212–261, 1994.

R. Lyons and Y. Peres. *Probability on Trees and Networks*. Cambridge University Press, 2012. In preparation. Current version available at `http://mypage.iu.edu/~rdlyons/`.

A. Maurer. Learning similarity with operator-valued large-margin classifiers. *Journal of Machine Learning Research*, 9:1049–1082, 2008.

S. Negahban and M. Wainwright. Restricted strong convexity and weighted matrix completion with noise. *Preprint*, 2010.

A.B. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, pages 615–622, 1962.

J. Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proc. Natl. Conf. on AI*, pages 133–136, 1982.

S. S. Rangapuram and M. Hein. Constrained 1-spectral clustering. In *Proc. 15th International Conference on Artificial Intelligence and Statistics*, AISTATS, 2012.

A. Rohde and A. Tsybakov. Estimation of high-dimensional low rank matrices. *Annals of Statistics*, 39(2):887–930, 2011.

S. Shalev-Shwartz, Y. Singer, and A. Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04. ACM, 2004.

K. Tsuda, G. Rätsch, and M. K. Warmuth. Matrix exponentiated gradient updates for on-line learning and bregman projections. *Journal of Machine Learning Research*, 6: 995–1018, 2005.

F. Vitale, N. Cesa-Bianchi, C. Gentile, and G. Zappella. See the tree through the lines: The shazoo algorithm. In *NIPS*, pages 1584–1592, 2011.

M. K. Warmuth. Winnowing subspaces. In *Proceedings of the 24th International Conference on Machine Learning*, pages 999–1006. ACM, 2007.

D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 296–303. ACM, 1996.

E. P. Xing, A. Y. Ng, M. I. Jordan, and J. Russell S. Distance metric learning with application to clustering with side-information. In *NIPS*, pages 505–512, 2002.

J. Zhang and R. Yan. On the value of pairwise constraints in classification and consistency. In *Proc of the 24th ICML*, ICML, 2007.

Appendix A. Proofs

This appendix contains all omitted proofs. Notation is as in the main text.

A.1. Missing proofs from Section 2

The set of example sequences *consistent* with a concept f for class prediction is denoted by $\mathcal{S}^c(f) := (\{(x, f(x))\}_{x \in \mathcal{X}})^*$, and for similarity prediction by $\mathcal{S}^s(f) := (\{((x', x''), \text{sim}(f(x'), f(x'')))\}_{x', x'' \in \mathcal{X}})^*$. A prediction algorithm is a mapping $A : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}$ from example sequences to prediction functions. Thus if A is a prediction algorithm and $S = (x_1, y_1), \ldots, (x_T, y_T) \in (\mathcal{X} \times \mathcal{Y})^*$ is an example sequence, then the *online* prediction mistakes are

$$M_A(S) := \sum_{t=1}^{T} [A((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))(x_t) \neq y_t] .$$

We write $S' \subseteq S$ to denote that S' is a subset of S as well to denote that S' is a *subsequence* of S.

We now introduce a weaker notion of a mistake bound as defined with respect to specific sequences rather than to the alternate notion of a concept. The weakness of this definition allows the construction in the following lemma to apply to "noisy" as well "consistent" example sequences.

Definition 9 *Given an algorithm A, we define the* **subsequential mistake bound** *with respect to an example sequence S as*

$$\mathbb{B}_A^\circ(S) := \max_{S' \subseteq S} M_A(S') .$$

Thus, a subsequential mistake bound is simply the "worst-case" mistake bound over all subsequences.

Lemma 10 *Given an online* classification *algorithm A, there exists a similarity algorithm A' such that for every sequence $S = (x_1, y_1), \ldots, (x_{2T}, y_{2T})$ its mistakes on*

$$S' = ((x_1, x_2), \text{sim}(y_1, y_2)), \ldots, ((x_{2T-1}, x_{2T}), \text{sim}(y_{2T-1}, y_{2T}))$$

is bounded as

$$M_{A'}(S')) \leq c \, \mathbb{B}_A^\circ(S) \log_2 K , \tag{6}$$

with $c < 5$.

Proof The proof of (6) works by a modification of the standard weighed majority algorithm (Littlestone and Warmuth, 1994) arguments. The key idea is that similarity reduces to classification if we received the actual class labels as feedback rather than just SIMILAR/DISSIMILAR as feedback. Since we do not have the class-labels, we instead "hallucinate" all possible feedback histories and then combine these histories on each trial using a weighted majority vote. If we only keep track of histories generated when the weighted majority vote is mistaken, the bound is small. Our master voting algorithm A' follows.

1. **Initialisation:** We initialize the parameter $\beta = 0.294$. We create a pool containing example sequences ("hallucinated *histories*") $\mathcal{S} := \{s\}$, with initially the empty history $s = \langle \rangle$ with weight $w_s := 1$.

2. **For** $t = 1, \ldots, T$ **do**

3. **Receive:** the pattern pair (x_{2t-1}, x_{2t})

4. **Predict:** SIMILAR if

$$\sum_{s \in \mathcal{S}} w_s [A(s)(x_{2t-1}) = A(s)(x_{2t})] \geq \sum_{s \in \mathcal{S}} w_s [A(s)(x_{2t-1}) \neq A(s)(x_{2t})]$$

otherwise predict DISSIMILAR.

5. **Receive:** Similarity feedback $\text{sim}(y_{2t-1}, y_{2t})$ if prediction was correct go to 2.

6. Two cases, first if this algorithm predicted SIMILAR when the pair was DISSIMILAR then for each history $s \in \mathcal{S}$ with a mistaken prediction create $K \times (K-1)$ histories $s_{1,2}, \ldots, s_{K,K-1}$ so that $s_{i,j}$ is equal to s but has the two "speculated" examples $(x_{2t-1}, i), (x_{2t}, j)$ appended to it. Then set $w_{s_{1,2}} = \ldots = w_{s_{K,K-1}} := \frac{\beta}{K(K-1)} w_s$ and remove s from $\mathcal{S}$. Second (predicted DISSIMILAR) as above but now we need to add only K new histories to the pool.

7. Go to 2.

Observe that there exists a history $s^* \in \mathcal{S}$ generated by no more $\mathbb{B}_A^\circ(S)$ "mistakes" since there is always at least one history in the pool $\mathcal{S}$ which is a subsequence of S. Thus

$$w_{s^*} \geq \left(\frac{\beta}{K(K-1)} \right)^{\mathbb{B}_A^\circ(S)} .$$

Furthermore, the total weight of the pool of histories $W := \sum_{s \in \mathcal{S}} w_s$ is reduced to a fraction of its weight no larger than $\frac{1+\beta}{2}$ whenever this master algorithm A' makes a mistake. Thus since $W \geq w_{s^*}$, we have

$$\left(\frac{1+\beta}{2} \right)^{M_{A'}(S')} \geq \left(\frac{\beta}{K(K-1)} \right)^{\mathbb{B}_A^\circ(S)} .$$

Solving for $M_{A'}(S')$ we have

$$M_{A'}(S') \leq \mathbb{B}_A^\circ(S) \left(\frac{\log_2 \frac{K(K-1)}{\beta}}{\log_2 \frac{2}{1+\beta}} \right) .$$

18

Substituting in $\beta = .294$ allows us to obtain the upper bound of (6) with $c \approx 4.99$. ∎

We observe, that reduction of similarity to classification holds for a wide variety online mistake bounds. Thus, e.g., we do not require the input sequence to be *consistent*, i.e., in S we may have examples (x', y') and (x'', y'') such that $x' = x''$ but $y' \neq y''$. The usual type of mistake bound is *permutation invariant* i.e., the bound is the "same" for all permutations of the input sequence; typical examples include the the WEIGHTED MAJORITY (Littlestone and Warmuth, 1994) and p-NORM PERCEPTRON (Grove et al., 2001; Gentile, 2003) algorithms. Observe that if $B_A^p(S)$ is a permutation invariant bound, then $\mathbb{B}_A^\circ(S) \leq B_A^p(S)$, since every subsequence of S is the prefix of a permutation of S. However, our reduction also more broadly applies to such "order-dependent" bounds, as the shifting-expert bounds in Herbster and Warmuth (1998).

We now show that classification reduces to similarity. This reduction is efficient and does not introduce a multiplicative constant but requires the stronger assumption of *consistency* not required by Lemma 10.

Lemma 11 *Given an online similarity algorithm A^s there exists an online classification algorithm A^c such that for any concept f if $S \in \mathcal{S}^c(f)$ then*

$$M_{A^c}(S) \leq \max_{S' \in \mathcal{S}^s(f)} M_{A^s}(S') + K \ . \tag{7}$$

Proof As a warm-up, pretend we know a set $P \subseteq X$ such that $|P| = K$ and for each $i \in \{1, \ldots, K\}$ there exists an $x \in P$ such that $f(x) = i$. Using P we create algorithm A^c as follows. We maintain a history (example sequence) h, which is initially empty. Then on every trial when we receive a pattern x_t we predict $\hat{y}_t \in \{f(x) : A^s(h)((x, x_t)) = \text{SIMILAR}, x \in P\}$, and if the set contains multiple elements or is empty then we predict arbitrarily. If A^c incurs a mistake, we add to our history h the K examples $((x, x_t), \text{sim}(f(x), f(x_t)))_{x \in P}$. Observe that if A^c incurs a mistake then at least one example corresponding to a mistaken similarity prediction is added to h and necessarily $h \in \mathcal{S}^s(f)$. Thus the mistakes of A^c are bounded by $\max_{S' \in \mathcal{S}^s(f)} M_{A^s}(S')$. Now, since we do not actually know a set P, we may modify our algorithm A^c so that although P is initially empty we predict as before, and if we make a mistake on x_t because there does not exist an $x \in P$ such that $f(x) = f(x_t)$, we then add x_t to P. We can only make K such mistakes, so we have the bound of (7). ∎

Proof of Theorem 1: If S is a classification sequence consistent with a concept f and A is a classification algorithm, then $\mathbb{B}_A^\circ(S) \leq \mathbb{B}_A(f)$, and hence (6) implies (1). Then, since (7) is equivalent to (2), we are done. □

A.1.1. THE $\log K$ TERM IS NECESSARY IN THEOREM 1

In our study of class prediction on graphs we observed (see Appendix A.2) that certain 2-class bounds may be converted to K-class bounds with no explicit dependence on K. Yet Theorem 1 introduces a factor of $\log K$ for similarity prediction. So a question that arises is this simply a byproduct of the above analysis or is the "$\log K$" factor tight. In the following,

we demonstrate it is tight by introducing the PAIRED PERMUTATION problem, which may be "solved" in the classification setting with no more than $\mathcal{O}(K)$ mistakes. Conversely, we show that an adversary can force $\Omega(K \log K)$ mistakes in the similarity setting.

We introduce the following notation. Let $z : \{1, \ldots, K\} \to \{1, \ldots, K\}$ denote a *permutation* function, a member of the set $\mathbb{Z}_K$ of all $K!$ bijective functions from $\{1, \ldots, K\}$ to $\{1, \ldots, K\}$. A *paired permutation* function is the mapping $y_z : \{1, \ldots, K\}^2 \to \{1, \ldots, K\}$, with $y_z(x', x'') := \max(z(x'), z(x''))$. So, for example, consider a 3-element permutation $z(1) \to 2, z(2) \to 3$, and $z(3) \to 1$. Then, e.g., $y_z(1, 2) \to 3$ and $y_z(1, 1) \to 2$. Thus, we define the set of the PAIRED PERMUTATION problem example sequences for class prediction as $\mathbb{PP}^c := \cup_{z \in \mathbb{Z}_K} \mathcal{S}^c(y_z)$, and for similarity as $\mathbb{PP}^s := \cup_{z \in \mathbb{Z}_K} \mathcal{S}^s(y_z)$.

Theorem 12 *There exists a class prediction algorithm A such that for any $S \in \mathbb{PP}^c$ we have $M_A(S) = \mathcal{O}(K)$. Furthermore, for any similarity prediction algorithm A', there exists an $S' \in \mathbb{PP}^s$ such that $M_{A'}(S') = \Omega(K \log K)$.*

Proof of Theorem 12: First, we show that there exists a class prediction algorithm A such that for any $S \in \mathbb{PP}^c$ we have $M_A(S) = \mathcal{O}(K)$. Consider the simpler problem for the concept class of permutations $\cup_{z \in \mathbb{Z}_K} \mathcal{S}^c(z)$. By simply predicting *consistently* with the past examples we cannot incur more than $K - 1$ mistakes. The algorithm $A_0(s)$ (CONSISTENT PREDICTOR), predicts y on receipt of pattern x_t if there exists some example (x, y) in its history s such that $x_t = x$, otherwise it predicts a $y \in \{1, \ldots, K\}$ not in its history. Now, using A_0 as a base algorithm, we can use the principle of the master algorithm of Lemma 10 to achieve $\mathcal{O}(K)$ mistakes for the paired permutation problem. Thus, when we receive a pair $((x', x''), y)$ either $y_z(x') = y$ or $y_z(x'') = y$, hence on a mistake we may "hallucinate" these two possible continuations. Our *class* prediction example sequence is $S = (x'_1, x''_1), y_1), \ldots, ((x'_T, x''_T), y_T) \in \mathbb{PP}^c$, and the algorithm A follows.

1. **Initialisation:** We initialize the parameter $\beta = 0.294$. We create a pool containing example sequences ("hallucinated *histories*") $\mathcal{S} := \{s\}$, with initially the empty history $s = \langle \rangle$ with weight $w_s := 1$.

2. **For** $t = 1, \ldots, T$ **do**

3. **Receive:** the pattern $x_t = (x'_t, x''_t)$

4. **Predict:**
$$\hat{y}_t = \operatorname*{argmax}_{k \in \{1, \ldots, K\}} \sum_{s \in \mathcal{S}} w_s [\max(A_0(s)(x'_t), A_0(s)(x''_t)) = k] . \tag{8}$$

5. **Receive:** Class feedback $y_t \in \{1, \ldots, K\}$. If prediction was correct go to 2.

6. For each history $s \in \mathcal{S}$ with a mistaken prediction, create two histories s', s'' so that s' (s'') is equal to s but has the example (x'_t, y) (the example (x''_t, y)) appended to it. Then set $w_{s'} = w_{s''} := \frac{\beta}{2} w_s$, and remove s from $\mathcal{S}$.

7. Go to 2.

Observe that there exists a history $s^* \in \mathcal{S}$ generated by no more $K - 1$ "mistakes". This is because, by induction, there is always a consistent history (i.e., the empty history is initially consistent, and when the master algorithm A makes a mistake and a consistent history makes a mistake, then either the continuation (x_t', y) or (x_t'', y) is consistent). Finally, observe that once a consistent history contains $K - 1$ examples, it can no longer make mistakes. Thus

$$w_{s^*} \geq \left(\frac{\beta}{2}\right)^{K-1}.$$

Furthermore, the total weight of the pool of histories $W := \sum_{s \in \mathcal{S}} w_s$ is reduced to a fraction of its weight no larger than $\frac{1+\beta}{2}$ whenever this master algorithm A makes a mistake. Since $W \geq w_{s^*}$, we have

$$\left(\frac{1+\beta}{2}\right)^{M_A(S)} \geq \left(\frac{\beta}{2}\right)^{K-1}.$$

Solving for $M_A(S)$ we can write

$$M_A(S) \leq (K-1)\left(\frac{\log_2 \frac{2}{\beta}}{\log_2 \frac{2}{1+\beta}}\right).$$

Substituting in $\beta = 0.294$ allows us to obtain the upper bound of $M_A(S) \leq 4.1(K-1)$. The argument then follows as in Theorem 1.

Now consider the similarity problem. If we receive an instance of the form $(((x', x''), (x'', x'')), y)$ (with $x' \neq x''$) then $y = \text{SIMILAR}$ implies $z(x') < z(x'')$ and $y = \text{DISIMILAR}$ implies $z(x') > z(x'')$. Thus with each mistaken example we learn precisely a single '<' comparison. It follows from standard lower bounds on comparison-based sorting algorithms (e.g., Cormen et al. (2001)) that an adversary can force $\Omega(K \log K)$ comparisons, and thus mistakes, for any "comparison"-algorithm to learn an arbitrary permutation. $\qquad\square$

Any problem associated with a set of example sequences $\mathcal{S}$ may be iterated into a set of r independent problems by a cross-product-like construction, so that if $S^1, \ldots, S^r \in \mathcal{S}$ and if $S^i = (x_1^i, y_1^i), \ldots, (x_{T_i}^i, y_{T_i}^i)$ then an r-iterated example sequence is

$$((x_1^1, 1), y_1^1), \ldots, ((x_{T_1}^1, 1), y_{T_1}^1), \ldots, ((x_1^i, i), y_1^i), \ldots, ((x_{T_r}^r, r), y_{T_r}^r).$$

We have simply conjoined the r example sequences into a single example sequence with each pattern "x" paired with an integer indicating from which sequence it originated. Thus by r-iterating the paired permutation problem we trivially observe mistake bounds of $\mathcal{O}(rK)$ and $\Omega(rK \log K)$ for all $r \in \mathbb{N}$ in the class and similarity setting, respectively, thereby implying that the multiplicative "$\log K$" gap occurs for an infinite family of classification/similarity problems.

A.2. Missing proofs from Section 3

LIFTING 2-CLASS PREDICTION TO K-CLASS PREDICTION ON GRAPHS

Suppose we have an algorithm for the 2-class graph labeling problem with a mistake bound of the form $M \leq \max(c|\Phi^G(\boldsymbol{y})|, 1)$ for all $\boldsymbol{y} \in \{1, 2\}^n$, for some $c \geq 1$, We show that this implies

the existence in the K-class setting of an algorithm with a bound of $M \leq \max(2c|\Phi^G(\boldsymbol{y})|, 1)$ for all $\boldsymbol{y} \in \{1, \ldots, K\}^n$, where K need not be known in advance to the algorithm.

The algorithm simply works by combining the predictions of "one versus rest" classifiers. We train one classifier per class, and introduce a new classifier as soon as that class first appears. On any given trial, the combination is straightforward: If there is only one classifier predicting with its own class then we go with that class, otherwise we just assume a mistake. Thus, on any given trial, we can only be mistaken if one of the current "one-verse-rest" classifiers makes a mistake. This implies that our mistake bound is the sum of the mistake bounds of all of the "one-verse-rest" classifiers. Because each such binary classifier has a mistake bound of the form $M \leq \max(c|\Phi_k^G(\boldsymbol{y})|, 1)$, and $\sum_{k=1}^{K} |\Phi_k^G(\boldsymbol{y})| = 2|\Phi^G(\boldsymbol{y})|$, we have that the K-class classifier has a bound of the form $\max(2c|\Phi^G(\boldsymbol{y})|, 1)$.

Proof of Theorem 3: We show that computing the partition function for the Ising model on a general graph reduces to computing the partition function problem for the Ising Model on a path graph with pairwise constraints hence showing #P-completeness. The partition problem for the Ising Model on a graph is defined by,

INSTANCE : An n-vertex graph G, and a natural number, β, presented in unary notation.

OUTPUT: The value of the partition function $Z_G(\beta)$,

$$Z_G(\beta) := \sum_{\boldsymbol{y} \in \{1,2\}^n} 2^{-\beta|\phi^G(\boldsymbol{y})|} .$$

This problem was shown #P-complete in (Jerrum and Sinclair, 1993, Theorem 15). The reduction to the partition problem on a path graph with constraints is as follows.

We are given a graph $G = (V_G, E_G)$ with $n = |V_G|$, and further assume each vertex is "labeled" uniquely from $1, \ldots, n$. We construct the following path graph with pairwise constraints (see Figure 4) for an illustration.

1. Find a spanning tree $T = (V_T, E_T)$ of G, and let $R = E_G - E_T$.

2. Perform a depth-first-visit of T. From the $2n - 1$ vertex visit sequence, create an isomorphic path graph P_0 with $2n - 1$ vertices such that each vertex in P_0 is labeled with the corresponding vertex label from the visit of T. Thus each edge of T is mapped to two edges in P_0.

3. We now proceed to create a path graph $P = (V_P, E_P)$ from P_0, which also includes each edge in R twice. We initialize P as a "duplicate" of P_0 including labels. For each edge $(v_r', v_r'') \in R$ we then do the following:

   (a) Choose an arbitrary vertex $v' \in V_P$ so that v' and v_r' have the same label;

   (b) Let v'''' be a *neighbor* of v' in P (i.e, $(v', v'''') \in E_P$);

   (c) Add vertices v'' and v''' to P with the labels of v_r'' and v_r', respectively;
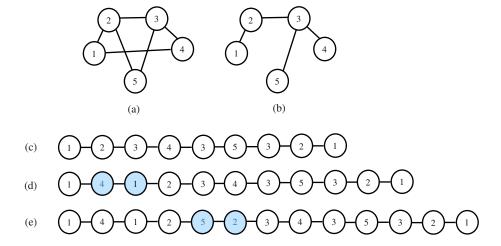
Figure 4: (a) The graph G with vertices labelled 1 to 5. (b) A spanning tree T of G. Note that $R = \{(1,4),(2,5)\}$. (c) The labeled path graph P_0. (d) Addition of vertices associated with edge (1,4) (the light blue vertices). (e) Addition of vertices associated with edge (2,5) (the light blue vertices), forming the path graph P. Note that every edge in G now has exactly two analogous edges in P, accounting for all the edges in P.

(d) Remove the edge (v', v'''') from P and add the edges $(v', v''), (v'', v''')$ and (v''', v'''') to P.

4. Finally create pairwise equality constraints between all vertices with the same label.

Thus observe for every edge in G there are two analogous edges in P, and furthermore if edge $(v, w) \notin G$ then there is not an analogous edge in P. Hence $Z_G(2\beta) = Z_P(\mathcal{C}, \beta)$.

Proof sketch Proposition 4 We start off with the Matrix Perceptron bound. For brevity, we write X_t instead of X_t^{p}. Also, let $\langle A, B \rangle$ be a shorthand for the inner product $\mathrm{TR}(A^T B)$. We can write

$$
\begin{aligned}
\langle X_t, X_t \rangle &= \mathrm{TR}(\sqrt{L^+}(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+}\sqrt{L^+}(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+}) \\
&= \mathrm{TR}((\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+}\sqrt{L^+}(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+}\sqrt{L^+}(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})) \\
&= ((\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T L^+ (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}))^2 \\
&= (R_{i_t,j_t}^G)^2 \\
&\leq R^2 \ .
\end{aligned}
$$

Moreover, for any $k \in \{1, \ldots, K\}$, define K vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K \in \mathbb{R}^n$ as follows.

$$
\boldsymbol{u}_k = (u_{k,1}, \ldots, u_{k,n})^\top, \qquad \text{with} \qquad u_{k,i} = [k = y_i] \ ,
$$

23

being y_i the label of the i-th vertex of G. Now, if we let $U := \sqrt{L} \left(\sum_{k=1}^{K} \boldsymbol{u}_k \boldsymbol{u}_k^\top \right) \sqrt{L}$, we have

$$
\begin{aligned}
\langle U, X_t \rangle &= \text{TR}(U^T X_t) \\
&= \sum_{k=1}^{K} \text{TR}(\sqrt{L} \boldsymbol{u}_k \boldsymbol{u}_k^T \sqrt{L} \sqrt{L^+} (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+}) \\
&= \sum_{k=1}^{K} \text{TR}((\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+} \sqrt{L} \boldsymbol{u}_k \boldsymbol{u}_k^T \sqrt{L} \sqrt{L^+} (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})) \\
&= \sum_{k=1}^{K} ((\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+} \sqrt{L} \boldsymbol{u}_k)^2 \ .
\end{aligned}
$$

By definition of pseudoinverse, $\sqrt{L}(\sqrt{L^+}\sqrt{L}\boldsymbol{u}_k) = (\sqrt{L}\sqrt{L^+}\sqrt{L})\boldsymbol{u}_k = \sqrt{L}\boldsymbol{u}_k$ for all $k = 1, \ldots, K$. Hence, $\sqrt{L^+}\sqrt{L}\boldsymbol{u}_k = \boldsymbol{u}_k + c\mathbf{1}$ for some $c \in \mathbb{R}$ as the constant vector $\mathbf{1}$ is in the null space of L. We therefore have that $(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T \sqrt{L^+}\sqrt{L}\boldsymbol{u}_k = u_{k,i_t} - u_{k,j_t}$, i.e.,

$$
\langle U, X_t \rangle = \sum_{k=1}^{K} (u_{k,i_t} - u_{k,j_t})^2 \ . \tag{9}
$$

Now, if $y_{i_t,j_t} = 0$ (i.e., $y_{i_t} = y_{j_t}$) then for all k we have $u_{k,i_t} - u_{k,j_t} = 0$, so that $\langle U, X_t \rangle = 0$. On the other hand, if $y_{i_t,j_t} = 1$ (i.e., $y_{i_t} \neq y_{j_t}$) then there exist distinct $a, b \in \{1, \ldots, K\}$ such that $|u_{a,i_t} - u_{a,j_t}| = |u_{b,i_t} - u_{b,j_t}| = 1$, and for all other $k \neq a, b$ we have $u_{k,i_t} - u_{k,j_t} = 0$. So, in this case $\langle U, X_t \rangle = 2$. This gives the linear separability condition of sequence $(X_1, y_{i_1,j_1}), (X_2, y_{i_2,j_2}), \ldots$ with respect to U.

Finally, we bound $\langle U, U \rangle$. Let $\Phi_{a,b}^G := \{(i,j) \in E : y_i = a, y_j = b\}$. We have:

$$
\begin{aligned}
\langle U, U \rangle &= \text{TR}(U^T U) \\
&= \text{TR}\left(\left(\sum_{a=1}^{K} \sqrt{L} \boldsymbol{u}_a \boldsymbol{u}_a^T \sqrt{L} \right) \left(\sum_{b=1}^{K} \sqrt{L} \boldsymbol{u}_b \boldsymbol{u}_b^T \sqrt{L} \right) \right) \\
&= \sum_{a=1}^{K} \sum_{b=1}^{K} \text{TR}(\sqrt{L} \boldsymbol{u}_a \boldsymbol{u}_a^T \sqrt{L} \sqrt{L} \boldsymbol{u}_b \boldsymbol{u}_b^T \sqrt{L}) \\
&= \sum_{a=1}^{K} \sum_{b=1}^{K} \text{TR}(\boldsymbol{u}_b^T \sqrt{L} \sqrt{L} \boldsymbol{u}_a \boldsymbol{u}_a^T \sqrt{L} \sqrt{L} \boldsymbol{u}_a) \\
&= \sum_{a=1}^{K} \sum_{b=1}^{K} (\boldsymbol{u}_b^T \sqrt{L} \sqrt{L} \boldsymbol{u}_a)^2 \\
&= \sum_{a=1}^{K} \sum_{b=1}^{K} (\boldsymbol{u}_b^T L \boldsymbol{u}_a)^2 \\
&= \sum_{a=1}^{K} \left(|\Phi_a^G|^2 + \sum_{b \neq a} |\Phi_{a,b}^G|^2 \right) \ .
\end{aligned}
$$

So, noticing that $\sum_{b\,:\,b\neq a}|\Phi^G_{a,b}| = |\Phi^G_a|$ and hence that $\sum_{b\,:\,b\neq a}|\Phi^G_{a,b}|^2 \leq |\Phi^G_a|^2$, we conclude that

$$\langle U, U\rangle \leq 2|\Phi^G|^2 \ .$$

With the above handy, the mistake bound on M^{P} easily follows from the standard analysis of the Perceptron algorithm with nonzero threshold.

By a similar token, the bound on M^{W} follows from the arguments in Warmuth (2007), after defining U to be a normalized version of the one we defined above for Matrix Perceptron, and noticing that X^{W}_t in Algorithm 1 are positive semidefinite and normalized to trace 1.

A.3. Missing proofs from Section 4

The following lemma relies on the equivalence between effective resistance $R^G_{i,j}$ of an edge (i,j) and its probability of being included in a randomly drawn spanning tree.

Lemma 13 *Let $(G, \boldsymbol{y})$ be a labeled graph, and T be a spanning tree of G drawn uniformly at random. Then, for all $k = 1, \ldots, K$, we have:*

1. $\mathbb{E}[|\Phi^T_k|] = \sum_{(i,j)\in\Phi^G_k} R^G_{i,j}$, and

2. $\mathbb{E}[|\Phi^T_k|^2] \leq 2(\sum_{(i,j)\in\Phi^G_k} R^G_{i,j})^2$.

Proof Set $s = |\Phi^G_k|$ and $\Phi^G_k = \{(i_1, j_1), (i_2, j_2), \ldots, (i_s, j_s)\}$. Also, for $\ell = 1, \ldots, s$, let X_ℓ be the random variable which is 1 if (i_ℓ, j_ℓ) is an edge of T, and 0 otherwise. From $\mathbb{E}[X_\ell] = R^G_{i_\ell, j_\ell}$ we immediately have 1). In order to prove 2), we rely on the negative correlation of variables X_ℓ, i.e., that $\mathbb{E}[X_\ell X_{\ell'}] \leq \mathbb{E}[X_\ell]\mathbb{E}[X_{\ell'}]$ for $\ell \neq \ell'$ (see, e.g., Lyons and Peres (2012)). Then we can write

$$
\begin{aligned}
\mathbb{E}(|\Phi^T_k|^2) &= \mathbb{E}\left[\left(\sum_{\ell=1}^{s} X_\ell\right)^2\right] \\
&= \mathbb{E}\left[\sum_{\ell=1}^{s}\sum_{\ell'=1}^{s} X_\ell X_{\ell'}\right] \\
&= \sum_{\ell=1}^{s}\mathbb{E}[X_\ell] + \sum_{\ell=1}^{s}\sum_{\ell'\neq\ell}\mathbb{E}[X_\ell X_{\ell'}] \\
&\leq \sum_{\ell=1}^{s}\mathbb{E}[X_\ell] + \sum_{\ell=1}^{s}\sum_{\ell'\neq\ell}\mathbb{E}[X_\ell]\mathbb{E}[X_{\ell'}] \ .
\end{aligned}
$$

Now, for any spanning tree T of G, if $s \geq 1$ then it must be the case that $|\Phi^T_k| \geq 1$, and hence $\sum_{\ell=1}^{s}\mathbb{E}[X_\ell] = \mathbb{E}[|\Phi^T_k|] \geq 1$. Combined with the above we obtain:

$$
\mathbb{E}[|\Phi^T_k|^2] \leq \left(\sum_{\ell=1}^{s}\mathbb{E}[X_\ell]\right)^2 + \sum_{\ell=1}^{s}\sum_{\ell'\neq\ell}\mathbb{E}[X_\ell]\mathbb{E}[X_{\ell'}] \leq 2\left(\sum_{\ell=1}^{s}\mathbb{E}[X_\ell]\right)^2 = 2\left(\sum_{\ell=1}^{s} R^G_{i_\ell, j_\ell}\right)^2 ,
$$

as claimed. ∎

Proof of Theorem 5 From Proposition 4 we have that if we execute Matrix Winnow on $B = B_T$ with similarity instances constructed via B, then the number M of mistakes satisfies

$$M = \mathcal{O}\left(|\Phi^B| \, D_B \log n\right) \ ,$$

where D_B is the resistance diameter of B. Since B is a tree, its resistance diameter is equal to its diameter, which is $\mathcal{O}(\log n)$. Moreover, $|\Phi_k^B| = \mathcal{O}(|\Phi_k^T| \log n)$, for $k = 1, \ldots, K$, hence $|\Phi^B| = \mathcal{O}(|\Phi^T| \log n)$. Plugging back, taking expectation over T, and using Lemma 13, 1) proves the Matrix Winnow bound. Similarly, if we run the Matrix Perceptron algorithm on B with similarity instances constructed via B then

$$M = \mathcal{O}\left(|\Phi^B|^2 \, D_B^2\right) \ .$$

Proceeding as before, in combination with Lemma 13, 2), proves the Matrix Perceptron bound.

Proof of Theorem 6 First of all, the fact that the algorithm is $\mathcal{O}(\log^2 n)$ per round easily follows from the fact that, since B is a balanced binary tree, the sizes of sets $\mathcal{P}_t$ (prediction step in (4)) and $\mathcal{S}_t$ (update step in (5)) are both $\mathcal{O}(\log n)$.

As for initialization time, a naive implementation would require $\mathcal{O}(n^2)$ (we must build the zero matrix F). We now outline a method of growing a data structure that stores a representation of F online for which the initialisation time is only $\mathcal{O}(n)$, while keeping the per round time to $\mathcal{O}(\log^2 n)$. For every vertex ℓ in B the algorithm maintains a subtree B_ℓ of B, initially set to $\{\rho\}$, being ρ the root of B. At every vertex $\ell' \in B_\ell$ is stored the value $F_{\ell,\ell'}$. At the start of time t, the algorithm climbs B from i_t to ρ, in doing so storing the ordered list $\mathcal{L}_{i_t}$ of vertices in the path from ρ to i_t. The same is done with j_t. The set $\mathcal{S}_t$ is then computed. For all $\ell \in \mathcal{S}_t$, the tree B_ℓ is then extended to include the vertices in $\mathcal{N}_t$ and the path from i_t (note that for each $\ell \in \mathcal{S}_t$ this takes only $\mathcal{O}(\log n)$ time, since we have the list $\mathcal{L}_{i_t}$). Whenever a new vertex ℓ' is added to B_ℓ, the value $F_{\ell,\ell'}$ is set to zero. Hence, we initialize F "on demand", the only initialization step being the allocation of the BST, i.e., $\mathcal{O}(n)$ time. We now continue by showing the equivalence of the sequence of predictions issued by (4) to those of the Matrix Perceptron algorithm with similarity instances constructed from B.

For every $\ell \in \mathcal{S}_t$ define $\Lambda_t(\ell)$ as the maximal subtree of B that contains ℓ and does not contain any nodes in $\mathcal{P}_t \setminus \{i_t, j_t\}$.

Lemma 14 $\Lambda_t(\cdot)$ *defined above enjoys the following properties (see Figure 5, left, for reference).*

1. *For all ℓ, $\Lambda_t(\ell)$ is uniquely defined;*

2. *Any subtree T of B that has no vertices from $\mathcal{P}_t \setminus \{i_t, j_t\}$ (and hence any of the trees Λ_t) contains at most one vertex from $\mathcal{S}_t$;*

3. *The subtrees $\{\Lambda_t(\ell) : \ell \in \mathcal{S}_t\}$ are pairwise disjoint;*

4. *The set $\{\Lambda_t(\ell) : \ell \in \mathcal{S}_t\} \cup (\mathcal{P}_t \setminus \{i_t, j_t\})$ covers B (so in particular $\{\Lambda_t(\ell) : \ell \in \mathcal{S}_t\}$ covers the set of leaves of B).*

Proof

1. Suppose we have subtrees T and T' with $T \neq T'$ that both satisfy the conditions of $\Lambda_t(\ell)$. Then w.l.o.g assume there exists a vertex ℓ' in T that is not in T'. Since T and T' are both connected and both contain ℓ, the subgraph $T \cup T'$ of B is connected and is hence a subtree. Since neither T nor T' contains vertices in $\mathcal{P}_t \setminus \{i_t, j_t\}$, $T \cup T'$ does not contain any such either. Hence, because T' is a strict subtree of $T \cup T'$, we have contradicted the maximality of T'.

2. Suppose T has distinct vertices $\ell, \ell' \in \mathcal{S}_t$. Since T is connected, it must contain the path in B from ℓ to ℓ'. This path goes from ℓ to the neighbor of ℓ that is in $\mathcal{P}_t \setminus \{i_t, j_t\}$, then follows the path $\mathcal{P}_t \setminus \{i_t, j_t\}$ (in the right direction) until a neighbor of ℓ' is reached. The path then terminates at ℓ'. Such a path contains at least one vertex in $\mathcal{P}_t \setminus \{i_t, j_t\}$, contradicting the initial assumption about T.

3. Assume the converse – that there exist distinct ℓ, ℓ' in $\mathcal{S}_t$ such that $\Lambda_t(\ell)$ and $\Lambda_t(\ell')$ share vertices. Then, since $\Lambda_t(\ell)$ and $\Lambda_t(\ell')$ are connected, $\Lambda_t(\ell) \cup \Lambda_t(\ell')$ must also be connected (and hence must be a subtree of B). Since $\Lambda_t(\ell) \cup \Lambda_t(\ell')$ shares no vertices with $\mathcal{P}_t \setminus \{i_t, j_t\}$, and contains both ℓ and ℓ' (which are both in $\mathcal{S}_t$), the statement in Item 2 above is contradicted.

4. Assume that we have a $\ell \in B \setminus (\mathcal{P}_t \setminus \{i_t, j_t\})$. Then let P' be the path from ℓ to the (first vertex encountered in) the path $\mathcal{P}_t \setminus \{i_t, j_t\}$. Let ℓ' be the second from last vertex in P'. Then ℓ' is a neighbor of a vertex in $\mathcal{P}_t$, but is not in $\mathcal{P}_t \setminus \{i_t, j_t\}$, so it must be in $\mathcal{S}_t$. This implies that the path P'' that goes from ℓ to ℓ' contains no vertices in $\mathcal{P}_t \setminus \{i_t, j_t\}$ and is therefore (Item 1) a subtree of $\Lambda_t(\ell')$. Hence, $\ell \in \Lambda_t(\ell')$. ∎

Lemma 15 *Let L be the Laplacian matrix of B, and $\ell, \ell' \in \mathcal{S}_t$. Then for any pair of vertices κ and κ' of B with $\kappa \in \Lambda_t(\ell)$ and $\kappa' \in \Lambda_t(\ell')$ we have*

$$(\boldsymbol{e}_\kappa - \boldsymbol{e}_{\kappa'})^T L^+ (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}) = f_t(\ell') - f_t(\ell) ,$$

where $\boldsymbol{e}_i$ is the i-th element in the canonical basis of $\mathbb{R}^{2n-1}$.

Proof We first extend the tagging function f_t to all vertices of B via the vector[6] $\tilde{f}_t$ as follows (note that, by Lemma 14, $\tilde{f}_t$ is well defined):

1. For all $\ell \in \mathcal{P}_t \setminus \{i_t, j_t\}$, set $\tilde{f}_t(\ell) = f_t(\ell)$;

2. For all $\ell' \in \mathcal{S}_t$ and $\ell \in \Lambda_t(\ell')$, set $\tilde{f}_t(\ell) = f_t(\ell')$.

---

6. In our notation, we interchangeably view $\tilde{f}$ both as a tagging function from the $2n-1$ vertices of B to the natural numbers and as a $(2n-1)$-dimensional vector.
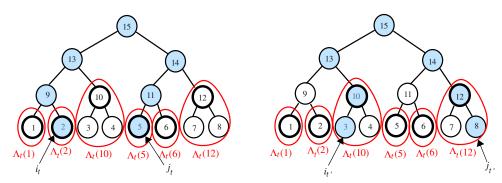
Figure 5: **Left:** The same BST as in Figure 2 with $i_t = 2$ and $j_t = 5$. Light blue vertices are those in $\mathcal{P}_t$. Thick-bordered vertices are those in $\mathcal{S}_t$. Since vertices 3 and 4 are in $\Lambda_t(10)$, we have $\tilde{f}_t(3) = \tilde{f}_t(4) = f_t(10)$. Since vertices 7 and 8 are in $\Lambda_t(12)$, we have $\tilde{f}_t(7) = \tilde{f}_t(8) = f_t(12)$. For all other vertices ℓ, we have $\tilde{f}_t(\ell) = f_t(\ell)$. **Right:** The same BST as in Figure 2 with path $\mathcal{P}_{t'}$ (light blue vertices) having endpoints $i_{t'} = 3$ and $j_{t'} = 8$. Thick-bordered vertices are still those in $\mathcal{S}_t$. Path $\mathcal{P}_{t'}$ intersects $\mathcal{S}_t$ at two vertices, 10 and 12, which means that $3 \in \Lambda_t(10)$ and $8 \in \Lambda_t(12)$. We have $\tilde{f}_t(3) = f_t(10)$, $\tilde{f}_t(8) = f_t(12)$, and $((e_3 - e_8)L^+(e_2 - e_5))^2 = (\tilde{f}_t(3) - \tilde{f}_t(8))^2 = (f_t(10) - f_t(12))^2$.

Claim 1 $L\tilde{f}_t = e_{j_t} - e_{i_t}$.

Proof of claim. For any vertex κ of $B \setminus \{i_t, j_t\}$ one of the following holds:

1. If $\kappa \in \mathcal{P}_t$, then κ has a neighbor κ_1 with $\tilde{f}_t(\kappa_1) = \tilde{f}_t(\kappa) - 1$, one neighbour κ_2 with $\tilde{f}_t(\kappa_2) = \tilde{f}_t(\kappa)+1$, and (unless κ is the root of B) one neighbour κ_3 with $\tilde{f}_t(\kappa_3) = \tilde{f}_t(\kappa)$. We therefore have that $[L\tilde{f}_t]_\kappa = 3\tilde{f}_t(\kappa) - \tilde{f}_t(\kappa_1) - \tilde{f}_t(\kappa_2) - \tilde{f}_t(\kappa_3) = 0$.

2. If $\kappa \in \mathcal{N}_t \setminus \mathcal{P}_t$, then κ has one neighbor κ_1 in $\mathcal{P}_t$ and we have $\tilde{f}_t(\kappa_1) = \tilde{f}_t(\kappa)$. Let T_κ be the subtree of B containing exactly vertex κ and all neighbors of κ bar κ_1. Since $\mathcal{P}_t$ is connected, it contains κ_1 and does not contain κ, none of the other neighbors of κ being in $\mathcal{P}_t$. Hence T_κ is a subtree of B that contains κ and no vertices from $\mathcal{P}_t \setminus \{i_t, j_t\}$, and so by Lemma 14, item 1 it must be a subtree of $\Lambda_t(\kappa)$. Hence, by definition of $\tilde{f}_t$, all vertices κ_2 in T_κ satisfy $\tilde{f}_t(\kappa_2) = \tilde{f}_t(\kappa)$. This implies that for all neighbors κ_3 of κ we have $\tilde{f}_t(\kappa_3) = \tilde{f}_t(\kappa)$, which in turn gives $[L\tilde{f}_t]_k = 0$.

3. If $\kappa \notin \mathcal{N}_t$ then, by Lemma 14 item 4, let κ be contained in $\Lambda_t(\ell)$ for some $\ell \in \mathcal{S}_t$. Let T_κ be the subtree of B containing exactly vertex κ and all neighbors of κ. Note that T_κ is a subtree of B that contains κ and no vertices from $\mathcal{P}_t \setminus \{i_t, j_t\}$. Since $\Lambda_t(\ell)$ also contains κ (hence $\Lambda_t(\ell) \cup T_\kappa$ is connected), we have that $\Lambda_t(\ell) \cup T_\kappa$ is a subtree of B that contains ℓ and no vertices from $\mathcal{P} \setminus \{i_t, j_t\}$. By Lemma 14 item 1, this implies that $\Lambda_t(\ell) \cup T_\kappa$ is a subtree of (and hence equal to) $\Lambda_t(\ell)$. Hence, by definition of $\tilde{f}_t$, we have that $\tilde{f}_t$ is identical on T_κ. Thus all neighbors κ_1 of κ satisfy $\tilde{f}_t(\kappa_1) = \tilde{f}_t(\kappa)$, implying again $[L\tilde{f}_t]_\kappa = 0$.

So in either case $[L\tilde{f}_t]_\kappa = 0$.

28

Finally, let i'_t be the neighbor of i_t in B. We have $[L\tilde{f}_t]_{i_t} = \tilde{f}_t(i_t) - \tilde{f}_t(i'_t) = 1 - 2 = -1$. Similarly, we have $[L\tilde{f}_t]_{j_t} = 1$. Putting together, we have shown that $L\tilde{f}_t = \boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}$, thereby concluding the proof of Claim 1.

Now, by definition of pseudoinverse,

$$L\tilde{f}_t = LL^+L\tilde{f}_t = LL^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}) \ .$$

This implies that $L(\tilde{f}_t - L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})) = 0$. Therefore there exists a constant c such that $\tilde{f}_t = L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t}) + c\mathbf{1}$. From the definition of $\tilde{f}$ we can write

$$
\begin{aligned}
f_t(\ell') - f_t(\ell) &= \tilde{f}_t(\kappa') - \tilde{f}_t(\kappa) \\
&= ([L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})]_{\kappa'} - c) - ([L^+(\boldsymbol{e}_{j_t} - \boldsymbol{e}_{i_t})]_{\kappa} - c) \\
&= (\boldsymbol{e}_\kappa - \boldsymbol{e}_{\kappa'})^T L^+(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}),
\end{aligned}
$$

as claimed. ∎

Lemma 16 *Let L be the Laplacian matrix of B, and κ, κ' be two vertices of B. Let $\mathcal{P}$ be the path from κ to κ' in B. Then for any t either $|\mathcal{P} \cap \mathcal{S}_t| \leq 1$ or $\mathcal{P} \cap \mathcal{S}_t = \{\ell, \ell'\}$, for two distinct vertices ℓ and ℓ'. No other cases are possible. Moreover,*

$$((\boldsymbol{e}_\kappa - \boldsymbol{e}_{\kappa'})^T L^+(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}))^2 = \begin{cases} 0 & \text{if } |\mathcal{P} \cap \mathcal{S}_t| \leq 1 \\ (f_t(\ell) - f_t(\ell'))^2 & \text{if } \mathcal{P} \cap \mathcal{S}_t = \{\ell, \ell'\} \end{cases} \ .$$

Proof By Lemma 14 item 4, we have two possible cases only:

1. There exists $\ell \in \mathcal{S}_t$ such that both κ and κ' are in $\Lambda_t(\ell)$: In this case (since $\Lambda_t(\ell)$ is connected) the path $\mathcal{P}$ lies in $\Lambda_t(\ell)$. Since, by Lemma 14 item 2, no $\ell' \in \mathcal{S}_t$ with $\ell' \neq \ell$ can be in $\Lambda_t(\ell)$, it is only ever possible that $\mathcal{P}$ contains at most one vertex ℓ (if any) of $\mathcal{S}_t$.

2. There exist two distinct nodes $\ell, \ell' \in \mathcal{S}_t$ such that $\kappa \in \Lambda(\ell)$ and $\kappa' \in \Lambda(\ell')$. In this case, $\mathcal{P}$ corresponds to the following path: First go from κ to ℓ (by Lemma 14 item 2, since this path lies in $\Lambda(\ell)$ the only vertex in $\mathcal{S}_t$ that lies in the section of the path is ℓ); then go to the neighbor of ℓ that is in $\mathcal{P}_t \setminus \{i_t, j_t\}$; then follow the path $\mathcal{P}_t \setminus \{i_t, j_t\}$ until you reach the neighbor of ℓ' (this section of $\mathcal{P}$ contains no vertices in $\mathcal{S}_t$); then go from ℓ' to κ (by Lemma 14 item 2, since this path lies in $\Lambda(\ell')$ the only vertex in $\mathcal{S}_t$ that lies in this section of the path is ℓ'). Thus, $\mathcal{P} \cap \mathcal{S}_t = \{\ell, \ell'\}$.

The result then follows by applying Lemma 15 to the two cases above. ∎

Figure 5 illustrates the above lemmas by means of an example.

To conclude the proof, let $\langle A, B \rangle$ be a shorthand for $\text{TR}(A^\top B)$. We see that from Algorithm 1, Lemma 16, and the definition of F in (4) we can write

$$
\begin{aligned}
\langle W_t, X_t \rangle &= \sum_{t'=1, t' \in \mathcal{M}}^{t-1} (2y_{i_{t'}, j_{t'}} - 1) \langle X_{t'}, X_t \rangle \\
&= \sum_{t'=1, t' \in \mathcal{M}}^{t-1} (2y_{i_{t'}, j_{t'}} - 1)((\boldsymbol{e}_{i_{t'}} - \boldsymbol{e}_{j_{t'}})^T L^+ (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t}))^2 \\
&= \sum_{(\ell, \ell') \in \mathcal{P}_t^2} F_{\ell, \ell'} \ ,
\end{aligned}
$$

where $\mathcal{M}$ is the set of mistaken rounds, and the second-last equality follows from a similar argument as the one contained in the proof of Proposition 4. Threshold $2 \log n$ in (4) is an upper bound on the radius squared $\langle X_t, X_t \rangle$ of instance matrices (denoted by R^2 in Algorithm 1). In fact, from the proof of Proposition 4,

$$
\max_t \langle X_t, X_t \rangle \le \max_{(i,j) \in V^2} ((\boldsymbol{e}_i - \boldsymbol{e}_j)^\top L^+ (\boldsymbol{e}_i - \boldsymbol{e}_j))^2 = \max_{(i,j) \in V^2} (R_{i,j}^G)^2 \ ,
$$

which is upper bounded by the square of the diameter $2 \log n$ of B.

A.4. Ancillary results, and missing proofs from Section 5

This section contains the proof of Theorem 7, along with preparatory results.

A digression on cuts and directed paths

Given[7] a connected and unweighted graph $G = (V, E)$, with $n = |V|$ vertices and $m = |E|$ edges, any partition of V into two subsets induces a *cut* over E. A cut is a *cutset* if it is induced by a two-connected component partition of V. Fix now any spanning tree T of G (this will be the one constructed by the algorithm at the end of the game, based on the paths produced by the adversary – see Section 5.1). In this context, the $n-1$ edges of T are often called *branches* and the remaining $m - n + 1$ edges are often called *chords*. Any branch of T cuts the tree into two components (it is therefore a cutset), and induces a two-connected component partition over V. Any such cutset is called a *fundamental cutset* of G (w.r.t. T). Cuts are always subsets of E, hence they can naturally be represented as (binary) indicator vectors with m components. For reasons that will be clear momentarily, it is also convenient to assign each edge an orientation (tail vertex to head vertex) and each cut an inward/outward direction. In particular, it is customary to give a fundamental cut the orientation of its branch. As a consequence of orientations/directions, cuts are rather represented as m-dimensional vectors whose components have values in $\{-1, 0, 1\}$.

Figure 6 (a) gives an example. In this figure, all edges are directed from the low index vertex to the high index vertex. Branch e_1 determines a cutset (more precisely, a fundamental cutset w.r.t. to the depicted spanning tree) separating vertices 2 and 7

---

7. The reader familiar with the theory of matroids will recognize what is recalled here as a well known example of a regular matroid on graphs. One can learn about them in standard textbooks/handbooks, e.g., (Gross and Yellen, 2003, Ch.6).
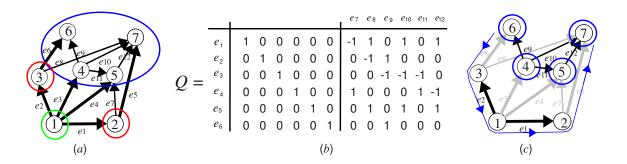
$$Q = \begin{array}{c|cccccc|cccccc}
 & & & & & & & e_7 & e_8 & e_9 & e_{10} & e_{11} & e_{12} \\
\hline
e_1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 1 \\
e_2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
e_3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 \\
e_4 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -1 \\
e_5 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
e_6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}$$

(a) (b) (c)

Figure 6: **(a)** A graph with oriented edges, tagged e_1 through e_{12}. A spanning tree T is denoted by the thick edges. The cutset $\{4, 5, 6, 7\}$ is depicted in blue. This cut has inward branches e_3, e_4, e_5, and e_6, and no outward branches. The red cutset separating vertex 3 from the rest has inward branch e_2, and outward branch e_6. Two other (nodal) cutsets are shown: one separating vertex 2 from the rest, and the other separating vertex 1 from the rest. **(b)** The fundamental cutset matrix associated with the chosen spanning tree. Viewed as a 12-dimensional vector, the blue cut is $q_{\{4,5,6,7\}} = (0, 0, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0)$, and can be represented as linear combination of rows Q_i of Q as $-Q_3 - Q_4 - Q_5 - Q_6$, i.e., by the vector of coefficients $\boldsymbol{u}_{\{4,5,6,7\}} = (0, 0, -1, -1, -1, -1)^\top$. Notice that chords e_7 and e_8 are both inward (coefficients -1 in $q_{\{4,5,6,7\}}$). **(c)** The connectivity structure induced by the selected spanning tree on the blue cutset in (a). For ease of reference, all edges in the blue cut have been turned into light gray. Vertices 4, 5, 6, and 7 are all connected in G under the blue cut, but are they are *all disconnected* in T. The path $6 \to 3 \to 1 \to 2 \to 7$ (depicted in blue) connects in T vertex 6 to vertex 7, and is represented by path $\boldsymbol{p} = (1, -1, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0)^\top$, hence $Q\boldsymbol{p} = (1, -1, 0, 0, 1, -1)^\top$. This path departs from the blue (disconnected) cluster $\{4, 5, 6, 7\}$ through edge e_6 (traversed "the wrong way") and returns to this set via e_5. Notice that $\boldsymbol{u}_{\{4,5,6,7\}}^\top Q\boldsymbol{p} = 0$.

from the remaining ones. Edge e_6 isolates just vertex 6 from the rest (again, a fundamental cutset). The fundamental cut determined by branch e_1 is represented by vector $q = (1, 0, 0, 0, 0, 0, -1, 1, 0, 1, 0, 1)$. This is because if we interpret the orientation of branch e_1 as outward to the cut, then e_7 is inward, e_8 is outward, as well as e_{10} and e_{12}. The matrix in Figure 6 (b) contains as rows all fundamental cutsets. This is usually called the *fundamental cutset matrix*, often denoted by Q (recall that this matrix depends on spanning tree T – for readability, we drop this dependence from our notation). Matrix Q has rank $n - 1$. Moreover, any cut (viewed as an m-dimensional vector) in the graph can be represented as a linear combination of fundamental cutset vectors with linear combination coefficients -1, $+1$, and 0. In essence, cuts are an $(n - 1)$-dimensional vector space with fundamental cutsets (rows Q_i of Q) as basis. It is important to observe that the vectors Q_i involved in this representation are precisely those corresponding to the branches of T that

are either moving inward (coefficient -1) or outward (coefficient $+1$). Hence the fewer are the branches of T cutting inward or outward, the sparser is this representation. Matrix Q has also further properties, like total unimodularity. This implies that any linear combination of their rows with coefficients in $\{-1, 0, +1\}$ will result in a vector whose coefficients are again in $\{-1, 0, +1\}$.

To summarize, given a spanning tree T of G, a direction for G's edges, and the associated matrix Q, any cutset[8] q in G can be represented as an m-dimensional vector $\boldsymbol{q} = Q^\top \boldsymbol{u}$, where $\boldsymbol{u} \in \{-1, 0, +1\}^{n-1}$ has as many nonzero components as are the branches of T belonging to q. With this representation (induced by T) in hand, we are essentially aimed at learning in a sequential fashion $\boldsymbol{u}$'s components.

In order to tie this up with our similarity problem, we view the edges belonging to a given cutset as the cut edges separating a (connected) cluster from the rest of the graph, and then associate with any given K-labeling of the vertices of G a sequence of K weight vectors $\boldsymbol{u}_k$, $k = 1, \ldots, K$, each one corresponding to one label. Since a given label can spread over multiple clusters (i.e., the vertices belonging to a given class label need not be a connected component of G), we first need to collect connected components belonging to the same cluster by summing the associated coefficient vectors. As an example, suppose in Figure 6 (a) we have 3 vertex labels corresponding to the three colors. The blue cluster contains vertices $4, 5, 6$ and 7, the green one vertex 1, and the red one vertices 2 and 3. Now, whereas the blue and the green labels are connected, the red one is not. Hence we have $\boldsymbol{u}_{\{4,5,6,7\}} = (0, 0, -1, -1, -1, -1)^\top$, $\boldsymbol{u}_{\{1\}} = (1, 1, 1, 1, 0, 0)^\top$, and $\boldsymbol{u}_{\{2,3\}} = (-1, -1, 0, 0, 1, 1)^\top$ is the sum of the two cutset coefficient vectors $\boldsymbol{u}_{\{2\}} = (-1, 0, 0, 0, 1, 0)^\top$, and $\boldsymbol{u}_{\{3\}} = (0, -1, 0, 0, 0, 1)^\top$. In general, our goal will then be to learn a sparse (and rank-K) matrix $U = \sum_{k=1}^K \boldsymbol{u}_k \boldsymbol{u}_k^\top$, where $\boldsymbol{u}_k$ corresponds to the k-th (connected or disconnected) class label.

Consistent with the above, we represent the pair of vertices (i_t, j_t) as an indicator vector encoding the unique path in T that connects the two vertices. This encoding takes edge orientation into account. For instance, the pair of vertices $(6, 7)$ in Figure 6 (c) is connected in T by path $p_{(6 \to 7)} = 6 \to 3 \to 1 \to 2 \to 7$. According to the direction of traversed edges (edge e_1 is traversed according to its orientation, edge e_2 in the opposite direction, etc.), path $p_{(6 \to 7)}$ is represented by vector $\boldsymbol{p} = (1, -1, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0)^\top = ((\boldsymbol{p}'_t)^\top | 0_{m-n+1}^\top)$, hence $Q\boldsymbol{p} = \boldsymbol{p}'_t = (1, -1, 0, 0, 1, -1)^\top$.[9] It is important to observe that computing $Q\boldsymbol{p}$ does not require full knowledge of matrix Q, since $Q\boldsymbol{p}$ only depends on T and the way its edges are traversed. With the above handy, we are ready to prove Theorem 7.

---

8. Though we are only interested in cutsets here, this statement holds more generally for any cut of the graph.

9. Any other path connecting 6 to 7 in G would yield the same representation. For instance, going back to Figure 6 (a), consider path $p'_{(6 \to 7)} = 6 \to 4 \to 7$, whose edges are not in T. This gives $\boldsymbol{p}' = (0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0)^\top$. Yet, $Q\boldsymbol{p}' = Q\boldsymbol{p} = (1, -1, 0, 0, 1, -1)^\top$. This invariance holds in general: Given the pair (i, j), the quantity $Q\boldsymbol{p}$ is independent of $\boldsymbol{p}$, if we let $\boldsymbol{p}$ vary over all paths in G departing from i and arriving at j. This common value is the $(n-1)$-dimensional vector containing the edges in the unique path in T joining i and j (taking traversal directions into account). Said differently, once we are given T, the quantity $Q\boldsymbol{p}$ only depends on i and j, not on the path chosen to connect them. This invariance easily follows from the fact that cuts are orthogonal to circuits, see, e.g., (Gross and Yellen, 2003, Ch.6).
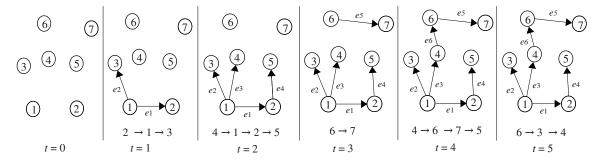
Figure 7: The way the r-norm Perceptron for similarity prediction of Section 5 builds instance vectors. At the beginning of the game ($t = 0$) the algorithm is only aware of the number of vertices ($n = 7$ in this case). In round $t = 1$ the pair $(2, 3)$ is generated, along with the connecting path $2 \to 1 \to 3$. Since none of the two revealed edges is creating cycles, the two edges $(1, 2)$ and $(1, 3)$ are added to the forest in this order. Hence, $\boldsymbol{x}_1 = (-1, 1, 0, 0, 0, 0)^\top$. Round $t = 2$: pair $(4, 5)$ and path $4 \to 1 \to 2 \to 5$ are disclosed. Edges $(1, 4)$ and $(2, 5)$ are revealed to the algorithm for the first time. The associated vector is then $\boldsymbol{x}_2 = (1, 0, -1, 1, 0, 0)^\top$. Round $t = 3$: A new edge is revealed which is disconnected from the previous subtree. We have $\boldsymbol{x}_3 = (0, 0, 0, 0, 1, 0)^\top$. Round $t = 4$: The algorithm receives pair $(4, 5)$ and corresponding path $4 \to 6 \to 7 \to 5$. While edge $(4, 6)$ is added to the forest, causing the two subtrees to merge, neither edge $(6, 7)$ nor edge $(7, 5)$ is added. In particular, $(7, 5)$ is not added because of the presence of an alternative path in the current forest (which is now a single tree) joining the two vertices. Hence the observed path $4 \to 6 \to 7 \to 5$ gets replaced by path $4 \to 1 \to 2 \to 5$, and the corresponding instance vector is $\boldsymbol{x}_4 = (1, 0, -1, 1, 0, 0)^\top$. Round $t = 5$: Since we have obtained a spanning tree, from this point on, no other edges will be added. In this round we have $\boldsymbol{x}_5 = (0, 0, 0, 0, 0, -1)^\top$, since the alternative (single edge) path $6 \to 4$ connecting 6 to 4 is already contained in the tree.

Proof of Theorem 7 For the constructed spanning tree[10] T, let $\boldsymbol{u}_k \in \{-1, 0, 1\}^{n-1}$ be the vector of coefficients representing the k-th class label w.r.t. the fundamental cutset matrix Q associated with T, and set $U = \sum_{k=1}^K \boldsymbol{u}_k \boldsymbol{u}_k^\top$. Also, let $\boldsymbol{x}_t$ be the instance vector computed by the algorithm at time t. Observe that, by the way $\boldsymbol{x}_t$ is constructed (see Figure 7 for an illustrative example) we have $\boldsymbol{x}_t = Q\boldsymbol{p}_t = \boldsymbol{p}'_t$ for all t, being $\boldsymbol{p}_t$ and $\boldsymbol{p}'_t$ the path vectors alluded at above. For any given class k, we have that $\boldsymbol{u}_k^\top \boldsymbol{x}_t = \boldsymbol{u}_k^\top \boldsymbol{p}'_t$. Recall that vector $\boldsymbol{u}_k$ contains $+1$ in each component corresponding to an outward branch of T, -1 in each

---

10. If less than $n - 1$ edges end up being revealed, the set of edges maintained by the algorithm cannot form a spanning tree of G. Hence T can be taken to be any spanning tree of G including all the revealed edges.

component corresponding to an inward branch, and 0 otherwise. We distinguish four cases (see Figure 6 (c), for reference):

1. i_t and j_t are both in the k-th class. In this case, the path in T that connects i_t to j_t must exit and enter the k-th class the same number of times (possibly zero). Since we only traverse branches, we have in the dot product $\boldsymbol{u}_k^\top \boldsymbol{p}_t'$ an equal number of $+1$ terms (corresponding to departures from the k-th class) and -1 (corresponding to arrivals). Hence $\boldsymbol{u}_k^\top \boldsymbol{p}_t' = 0$. Notice that this applies even when the k-th class is not connected.

2. i_t is in the k-th class, but j_t is not. In this case, the number of departures from the k-th class should exceed the number of arrivals by exactly one. Hence we must have $\boldsymbol{u}_k^\top \boldsymbol{p}_t' = 1$.

3. i_t is not in the k-th class, but j_t is. By symmetry (swapping i_t with j_t), we have $\boldsymbol{u}_k^\top \boldsymbol{p}_t' = -1$.

4. Neither i_t nor j_t is in the k-th class. Again, we have an equal number of arrival/departures to/from the k-th class (possibly zero), hence $\boldsymbol{u}_k^\top \boldsymbol{p}_t' = 0$.

We are now in a position to state our linear separability condition. We can write

$$\text{VEC}(U)^\top \text{VEC}(X_t) = \text{TR}(U^\top X_t) = \text{TR}(\sum_{k=1}^K \boldsymbol{u}_k \boldsymbol{u}_k^\top \boldsymbol{x}_t \boldsymbol{x}_t^\top) = \sum_{k=1}^K (\boldsymbol{u}_k^\top \boldsymbol{x}_t)^2 = \sum_{k=1}^K (\boldsymbol{u}_k^\top \boldsymbol{p}_t')^2,$$

which is 2 if i_t and j_t are in different classes (i.e., i_t and j_t are dissimilar), and 0, otherwise (i.e., i_t and j_t are similar). We have therefore obtained that the label y_t associated with (i_t, j_t) is delivered by the following linear-threshold function:

$$y_t = \begin{cases} 1 & \text{if } \text{VEC}(U)^\top X_t \geq 1 \\ 0 & \text{otherwise}. \end{cases} \tag{10}$$

Because we can interchangeably view $\text{VEC}(\cdot)$ as vectors or matrices, this opens up the possibility of running any linear-threshold learning algorithm (on either vectors or matrices). For r-norm Perceptrons with the selected norm r and decision threshold, we have a bound on the number M of mistakes of the form (Grove et al., 2001; Gentile, 2003)

$$M = \mathcal{O}\left(||\text{VEC}(U)||_1^2 \, ||\text{VEC}(X_t)||_\infty^2 \, \log n\right) ,$$

where

$$||\text{VEC}(U)||_1 = ||\text{VEC}(\sum_{k=1}^K \boldsymbol{u}_k \boldsymbol{u}_k^\top)||_1 = ||\sum_{k=1}^K \text{VEC}(\boldsymbol{u}_k \boldsymbol{u}_k^\top)||_1 \leq \sum_{k=1}^K ||\text{VEC}(\boldsymbol{u}_k \boldsymbol{u}_k^\top)||_1 = \sum_{k=1}^K ||\boldsymbol{u}_k||_1^2 ,$$

and

$$||\text{VEC}(X_t)||_\infty = ||\text{VEC}(\boldsymbol{x}_t \boldsymbol{x}_t^\top)||_\infty = ||\boldsymbol{x}_t||_\infty^2 = 1 .$$

Moreover, by the way vectors $\boldsymbol{u}_k$ are constructed, we have $||\boldsymbol{u}_k||_1 = |\Phi_k^T|$. In turn, $|\Phi_k^T| \leq |\Phi_k^G|$ holds independent of the connectedness of the k-th cluster. Putting together and upper bounding concludes the proof.