Random Spanning Trees and the Prediction of Weighted Graphs

Nicolò Cesa-Bianchi CESA-BIANCHI@DSI.UNIMI.IT
DSI, Università di Milano, Italy

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

Fabio Vitale FABIO.VITALE@UNIMI.IT
DSI, Università di Milano, Italy

Giovanni Zappella GIOVANNI.ZAPPELLA@STUDENTI.UNIMI.IT
DSI, Università di Milano, Italy

Abstract

We show that the mistake bound for predicting the nodes of an arbitrary weighted graph is characterized (up to logarithmic factors) by the cutsize of a random spanning tree of the graph. The cutsize is induced by the unknown adversarial labeling of the graph nodes. In deriving our characterization, we obtain a simple randomized algorithm achieving the optimal mistake bound on any weighted graph. Our algorithm draws a random spanning tree of the original graph and then predicts the nodes of this tree in constant amortized time and linear space. Experiments on real-world datasets show that our method compares well to both global (Perceptron) and local (label-propagation) methods, while being much faster.

1. Introduction

A widespread approach to the solution of classification problems is representing the data through a weighted graph in which edge weights quantify the similarity between data points. This technique for coding input data has been applied to several domains, including Web spam detection (Herbster et al., 2009b), classification of genomic data (Tsuda & Schölkopf, 2009), face recognition (Chang & Yeung, 2006), and text categorization (Goldberg & Zhu, 2004). In most applications, edge weights are computed through a complex data-modelling process and convey crucially important information for classifying nodes.

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

This paper focuses on the online version of the graph classification problem: The entire graph is known in advance and, at each step, the algorithm is required to predict the label of a new arbitrarily chosen node. In the special case of unweighted graphs (where all edges have unit weight) a key parameter for controlling the number of prediction mistakes is the size of the cut induced by the unknown adversarial labeling of the nodes. Although in the unweighted case previous studies use the cutsize to prove several interesting upper bounds (Herbster et al., 2009a; Herbster & Pontil, 2007; Herbster et al., 2009b), no general lower bounds on the number of prediction mistakes are known, leaving fully open the question of characterizing the complexity of learning a labeled graph. In a recent paper, Cesa-Bianchi et al. (2009) bound the expected number of mistakes by the cutsize of a random spanning tree of the graph, a quantity typically much smaller than the cutsize of the whole graph. In this paper we show that this quantity captures the hardness of the graph learning problem. Given any weighted graph, we prove that any prediction algorithm must err on a number of nodes which is at least as big as the cutsize of the graph's random spanning tree (which is defined in terms on the graph's weights). Moreover, we exhibit a simple algorithm achieving the optimal mistake bound to within logarithmic factors on any sufficiently connected weighted graph whose weighted cutsize is not an overwhelming fraction of the total weight.

Following Cesa-Bianchi et al. (2009), our algorithm first extracts a random spanning tree of the original graph. Then, it predicts all nodes of this tree using a generalization of the method proposed by Herbster et al. (2009a). Our tree prediction procedure is extremely efficient: it only requires *constant* amortized time per prediction and space *linear in the number of*

nodes. Note that computational efficiency is a central issue in practical applications where the involved datasets can be very large. In such contexts, learning algorithms whose time complexity scales, say, more than quadratically with the number of data points should be considered impractical.

A further significant contribution of this work is the experimental evaluation of our method, as compared to methods recently proposed in the literature on graph prediction. In particular, we compare our algorithm to the Perceptron algorithm with Laplacian kernel by Herbster & Pontil (2007); Herbster et al. (2009b), and to the label propagation algorithm by Zhu et al. (2003), including its online version. The experiments have been carried out on four medium-sized real-world datasets. The two tree-based algorithms (ours and the Perceptron algorithm) have been tested using spanning trees generated in various ways. Though not yet conclusive, our experimental comparison, shows that our online algorithm compares well to all competitors while using, in most cases, the least amount of time and memory resources.[1]

2. Preliminaries and basic notation

Let $G = (V, E, W)$ be an undirected, connected, and weighted graph with n nodes and positive edge weights $w_{i,j} > 0$ for $(i, j) \in E$. A labeling of G is any assignment $\boldsymbol{y} = (y_1, \ldots, y_n) \in \{-1, +1\}^n$ of binary labels to its nodes. We use $(G, \boldsymbol{y})$ to denote the resulting labeled weighted graph. The online learning protocol for predicting $(G, \boldsymbol{y})$ is defined as follows. The learner is given G while $\boldsymbol{y}$ is kept hidden. The nodes of G are presented to the learner one by one, according to an unknown and arbitrary permutation $i_1, \ldots, i_n$ of V. At each time step $t = 1, \ldots, n$ node i_t is presented and the learner must predict its label y_{i_t}. Then y_{i_t} is revealed and the learner knows whether a mistake occurred. The learner's goal is to minimize the total number of prediction mistakes.

It is reasonable to expect that prediction performance should degrade with the increase of "randomness" in the labeling. For this reason, our analysis of graph prediction algorithms bounds from above the number of prediction mistakes in terms of appropriate notions of graph label *regularity*. A standard notion of label regularity is the cutsize of a labeled graph, defined as follows. A ϕ-edge of a labeled graph $(G, \boldsymbol{y})$ is any edge (i, j) such that $y_i \neq y_j$. Similarly, an edge (i, j) is ϕ-free if $y_i = y_j$. Let $E^\phi \subseteq E$ be the set of ϕ-edges in $(G, \boldsymbol{y})$. The *cutsize* $\Phi_G(\boldsymbol{y})$ of $(G, \boldsymbol{y})$ is the number of ϕ-edges in E^ϕ, i.e., $\Phi_G(\boldsymbol{y}) = \left|E^\phi\right|$ (independent of the edge weights). The *weighted* cutsize $\Phi_G^W(\boldsymbol{y})$ of $(G, \boldsymbol{y})$ is $\Phi_G^W(\boldsymbol{y}) = \sum_{(i,j)\in E^\phi} w_{i,j}$.

Fix $(G, \boldsymbol{y})$. Let $r_{i,j}^W$ be the effective resistance between nodes i and j of G. In the interpretation of the graph as an electric network where the weights $w_{i,j}$ are the edge conductances, the effective resistance $r_{i,j}^W$ is the voltage between i and j when a unit current flow is maintained through them. For $(i, j) \in E$, let also $p_{i,j} = w_{i,j} r_{i,j}^W$ be the probability that (i, j) belongs to a random spanning tree T —see, e.g., (Lyons & Peres, 2009). Then we have

$$\mathbb{E}\,\Phi_T(\boldsymbol{y}) = \sum_{(i,j)\in E^\phi} p_{i,j} = \sum_{(i,j)\in E^\phi} w_{i,j} r_{i,j}^W \qquad (1)$$

where the expectation $\mathbb{E}$ is over the random choice of spanning tree T. Since $\sum_{(i,j)\in E} p_{i,j}$ is equal to $n - 1$, irrespective of the edge weighting, we have $0 \leq \mathbb{E}\,\Phi_T(\boldsymbol{y}) \leq n - 1$. Hence the ratio $\frac{1}{n-1}\mathbb{E}\,\Phi_T(\boldsymbol{y}) \in [0, 1]$ provides a *density-independent* measure of the cutsize in G, and even allows to compare labelings on different graphs. It is also important to note that $\mathbb{E}\,\Phi_T(\boldsymbol{y})$ can be much smaller than $\Phi_G^W(\boldsymbol{y})$ when there are strongly connected regions in G contributing prominently to the weighted cutsize. To see this, consider the following scenario: If $(i, j) \in E^\phi$ and $w_{i,j}$ is large, then (i, j) gives a big contribution to[2] $\Phi_G^W(\boldsymbol{y})$. However, this might not happen in $\mathbb{E}\,\Phi_T(\boldsymbol{y})$. In fact, if i and j are strongly connected (i.e., if there are many disjoint paths connecting them), then $r_{i,j}^W$ is very small and the terms $w_{i,j} r_{i,j}^W$ in (1) are small too. Therefore, the effect of the large weight $w_{i,j}$ may often be compensated by the small probability of including (i, j) in the random spanning tree.

3. A general lower bound

We start by stating a general lower bound, showing that any prediction algorithm must err at least $\frac{1}{2}\mathbb{E}\,\Phi_T(\boldsymbol{y})$ times on any weighted graph.

Theorem 1 *Let $G = (V, E, W)$ be a weighted undirected graph with n nodes and weights $w_{i,j} > 0$ for $(i, j) \in E$. Then for all $K \leq n$ there exists a randomized labeling $\boldsymbol{y}$ of G such that for all (deterministic or randomized) algorithms A, the expected number of prediction mistakes made by A is at least $K/2$, while $\mathbb{E}\,\Phi_T(\boldsymbol{y}) < K$.*

[1] Due to space limitations, all proofs are omitted from this extended abstract. The omitted material can be found in the longer version (Cesa-Bianchi et al., 2010).

[2] It is easy to see that in such cases $\Phi_G^W(\boldsymbol{y})$ can be much larger than n.

4. The Weighted Tree Algorithm

We now describe the Weighted Tree Algorithm (WTA) for predicting the labels of a weighted tree. In Section 5 we show how to apply WTA to the more general weighted graph prediction problem. WTA first turns the tree into a line graph (i.e., a list), then runs a fast nearest neighbor method to predict the labels of each node in the line. Though this technique is similar to that one used in (Herbster et al., 2009a), the fact that the tree is weighted makes the analysis significantly more difficult, and the practical scope of our algorithm significantly wider. Our experimental comparison in Section 7 confirms that exploiting the weight information is often beneficial in graph prediction.

Given a labeled weighted tree $(T, \boldsymbol{y})$, the algorithm initially creates a weighted line graph L' containing some duplicates of the nodes in T. Then, each duplicate node (together with its incident edges) is replaced by a single edge with a suitably chosen weight. This results in the final weighted line graph L which is then used for prediction. In order to create L from T, WTA performs the following *tree linearization* steps:

1. An arbitrary node r of T is chosen, and a line L' containing only r is created.
2. Starting from r, a depth-first visit of T is performed. Each time an edge (i, j) is traversed (even in a backtracking step), the edge is appended to L' with its weight $w_{i,j}$, and j becomes the current terminal node of L'. Note that backtracking steps can create in L' at most one duplicate of each edge in T, while nodes in T may be duplicated several times in L'.
3. L' is traversed once, starting from terminal r. During this traversal, duplicate nodes are eliminated as soon as they are encountered. This works as follows. Let j be a duplicate node, and (j', j) and (j, j'') be the two incident edges. The two edges are replaced by a new edge (j', j'') having weight[3] $w_{j',j''} = \min\{w_{j',j}, w_{j,j''}\}$. Let L be the resulting line.

The analysis of Section 4.1 shows that this choice of $w_{j',j''}$ guarantees that the weighted cutsize of L is smaller than twice the weighted cutsize of T. Once L is created from T, the algorithm predicts the label of each node i_t using a nearest-neighbor rule operating on L with a *resistance distance* metric. That is, the prediction on i_t is the label of i_{s^*}, being $s^* = \operatorname{argmin}_{s<t} d(i_s, i_t)$ the previously revealed node closest to i_t, and $d(i, j) = \sum_{s=1}^{k} 1/w_{v_s,v_{s+1}}$ is the sum of the resistors (i.e., reciprocals of edge weights) along the unique path $i = v_1 \to v_2 \to \cdots \to v_{k+1} = j$ connecting node i to node j.

[3] By iterating this elimination procedure, it might happen that more than two adjacent nodes get eliminated. In this case, the two surviving terminal nodes are connected in L by the lightest edge among the eliminated ones in L'.

4.1. Analysis of WTA

The following lemma gives a mistake bound on WTA run on any weighted line graph. Let $R_G^W = \sum_{(i,j)\in E\setminus E^\phi} 1/w_{i,j}$ the sum of resistors of ϕ-free edges in a labeled graph $(G, \boldsymbol{y})$. Let also $f \overset{\mathcal{O}}{=} g$ denote $f = \mathcal{O}(g)$.

Lemma 2 *If* WTA *is run on a weighted line graph* $(L, \boldsymbol{y})$*, then the total number* m_L *of mistakes satisfies*

$$m_L \overset{\mathcal{O}}{=} \Phi_L(\boldsymbol{y}) \left(1 + \log\Big(1 + \frac{\widetilde{R}_L^W \Phi_L^W(\boldsymbol{y})}{\Phi_L(\boldsymbol{y})}\Big)\right) + K$$

where $\widetilde{R}_L^W$ *is the sum of the resistors of any arbitrary set including all but* K ϕ*-free edges of* L*.*

Note that the bound of Lemma 2 implies that, for any $K \geq 0$, one can drop from the bound the contribution of any set of K resistors in R_T^L at the cost of adding K extra mistakes. We now provide an upper bound on the number of mistakes made by WTA on any weighted tree $T = (V, E, W)$ in terms of the number of ϕ-edges, the weighted cutsize, and R_T^W.

Theorem 3 *If* WTA *is run on a weighted and labeled tree* $(T, \boldsymbol{y})$*, then the total number* m_T *of mistakes satisfies*

$$m_T \overset{\mathcal{O}}{=} \Phi_T(\boldsymbol{y}) \left(1 + \log\left(1 + \frac{R_T^W \Phi_T^W(\boldsymbol{y})}{\Phi_T(\boldsymbol{y})}\right)\right) .$$

The logarithmic factor in the above bound shows that the algorithm takes advantage of labelings such that the weights of ϕ-edges are small (thus making $\Phi_T^W(\boldsymbol{y})$ small) and the weights of ϕ-free edges are high (thus making R_T^W small). This somehow matches the intuition behind WTA's nearest-neighbor rule according to which nodes that are close to each other are expected to have the same label. In particular, observe that the way the above quantities are combined makes the bound independent of rescaling of the edge weights. Again, this has to be expected, since WTA's prediction is scale insensitive. On the other hand, it may appear less natural that the mistake bound also depends linearly on the cutsize $\Phi_T(\boldsymbol{y})$, *independent of the edge weights*. As a matter of fact, this linear dependence on the unweighted cutsize *cannot be eliminated* (this is a simple consequence of Theorem 1 in Section 3).

5. Predicting a weighted graph

In order to solve the more general problem of predicting the labels of a weighted graph G, one can first

generate a spanning tree T of G and then run WTA directly on T. In this case it is possible to rephrase Theorem 3 in terms of properties of G. Note that for each spanning tree T of G, $\Phi_T^W(\boldsymbol{y}) \leq \Phi_G^W(\boldsymbol{y})$ and $\Phi_T(\boldsymbol{y}) \leq \Phi_G(\boldsymbol{y})$. Specific choices of the spanning tree T control in different ways the quantities in the mistake bound of Theorem 3. For example, a minimum spanning tree tends to reduce the value of R_T^W, betting on the fact that ϕ-edges are light. The next theorem relies on *random* spanning trees.

Theorem 4 *If* WTA *is run on a random spanning tree T of a labeled weighted graph $(G, \boldsymbol{y})$, then the total number m_G of mistakes satisfies*

$$\mathbb{E}\, m_G \overset{\mathcal{O}}{=} \mathbb{E}\big[\Phi_T(\boldsymbol{y})\big]\Big(1 + \log\Big(1 + w_{\max}^{\phi}\mathbb{E}\big[R_T^W\big]\Big)\Big) \quad (2)$$

where $w_{\max}^{\phi} = \max_{(i,j)\in E^{\phi}} w_{i,j}$.

Note that the mistake bound in (2) is scale-invariant, since $\mathbb{E}\big[\Phi_T(\boldsymbol{y})\big] = \sum_{(i,j)\in E^{\phi}} w_{i,j} r_{i,j}^W$ cannot be affected by a uniform rescaling of the edge weights, and so is the product $w_{\max}^{\phi}\mathbb{E}\big[R_T^W\big] = w_{\max}^{\phi}\sum_{(i,j)\in E\setminus E^{\phi}} r_{i,j}^W$.

We now compare the mistake bound (2) to the lower bound stated in Theorem 1. In particular, we prove that WTA is optimal (up to $\mathcal{O}(\log n)$ factors) on every weighted connected graph in which the ϕ-edges weights are not "superpolynomially overloaded" w.r.t. the ϕ-free edge weights. In order to rule out pathological cases, when the weighted graph is nearly disconnected, we impose the following mild assumption on the graphs being considered.

We say that a graph is **polynomially connected** if the ratio of any pair of effective resistances (even those between nonadjacent nodes) in the graph is polynomial in the total number of nodes n. This definition essentially states that a weighted graph can be considered connected if no pair of nodes can be found which is substantially less connected than any other pair of nodes. Again, as one would naturally expect, this definition is independent of uniform weight rescaling. The following corollary shows that if WTA is not optimal on a polynomially connected graph, then the labeling must be so irregular that the total weight of ϕ-edges is an overwhelming fraction of the overall weight.

Corollary 5 *Pick any polynomially connected weighted graph G with n nodes. If the ratio of the total weight of ϕ-edges to the total weight of ϕ-free edges is bounded by a polynomial in n, then the total number of mistakes m_G made by* WTA *when run on a random spanning tree T of G satisfies* $\mathbb{E}\, m_G \overset{\mathcal{O}}{=} \mathbb{E}\big[\Phi_T(\boldsymbol{y})\big]\log n$.

Note that when the hypothesis of this corollary is not satisfied the bound of WTA is not necessarily vacuous. For example, $\mathbb{E}\big[R_T^W\big]w_{\max}^{\phi} = n^{\mathrm{polylog}(n)}$ implies an upper bound which is optimal up to polylog(n) factors. In particular, having a constant number of ϕ-free edges with exponentially large resistance contradicts the assumption of polynomial connectivity, but it need not lead to a vacuous bound in Theorem 4. In fact, one can use Lemma 2 to drop from the mistake bound of Theorem 4 the contribution of any set of $\mathcal{O}(1)$ resistances in $\mathbb{E}\big[R_T^W\big] = \sum_{(i,j)\in E\setminus E^{\phi}} r_{i,j}^W$ at the cost of adding just $\mathcal{O}(1)$ extra mistakes. This could be interpreted as a robustness property of WTA's bound against graphs that do not fully satisfy the connectedness assumption.

Corollary 5 can be compared to the expected mistake bound of the graph Perceptron algorithm GPA on the same random spanning tree —see Section 7 for more details on GPA. This bound depends on the expectation of the product $\Phi_T^W(\boldsymbol{y})D_T^W$, where D_T^W is the diameter of T in the resistance distance metric. Note that these two factors are negatively correlated because $\Phi^W(\boldsymbol{y})$ dipends linearly on the edge weights whereas D_T^W dipends linearly on the reciprocal of these weights —see the definition of resistance distance in Section 2. Moreover, for any given scale of the edge weights, D_T^W can be linear in the number n of nodes.

6. Implementation

A direct implementation of WTA operating on a tree T with n nodes runs in time $\mathcal{O}(n \log n)$ and requires linear memory space. We now describe how to implement WTA to run in time $\mathcal{O}(n)$, i.e., in *constant* amortized time per step.

Once the given tree T is linearized into an n-node line L, we initially traverse L from left to right. Call j_0 the left-most terminal node of L. During this traversal, the resistance distance $d(j_0, i)$ is incrementally computed for each node i in L. This makes it possible to calculate $d(i, j)$ in constant time for any pair of nodes, since $d(i, j) = |d(j_0, i) - d(j_0, j)| \;\; \forall i, j \in L$. On top of line L a complete binary tree T' with $2^{\lceil \log_2 n \rceil}$ leaves is constructed.[4] The k-th leftmost leaf (in the usual tree representation) of T' is the k-th node in L (numbering the nodes of L from left to right). The algorithm maintains this data-structure in such a way that at time t: (i) the subsequence of leaves whose labels are revealed at time t are connected through a (bidirectional) list B, and (ii) all the ancestors in T' of the leaves of B are marked. See Figure 1.

When WTA is required to predict the label y_{i_t}, the algorithm looks for the two closest leaves i' and i''

[4] For simplicity, this description assumes n is a power of 2. If this is not the case, we could add dummy nodes to L before building T'.

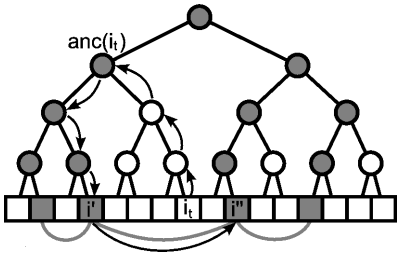

Figure 1. Constant amortized time implementation of WTA. The line L has $n = 16$ nodes (the adjacent squares at the bottom). Shaded squares are the revealed nodes, connected through a dark grey doubly-linked list B. The depicted tree T' has both unmarked (white) and marked (shaded) nodes. The arrows indicate the traversal operations performed by WTA when predicting the label of node i_t: The upward traversal stops as soon as a marked ancestor $\mathrm{anc}(i_t)$ is found, and then a downward traversal begins. Note that WTA first descends to the left, and then keeps going right all the way down. Once i' is determined, a single step within B suffices to determine i''.

oppositely located in L with respect to i_t. The above data structure supports this operation as follows. WTA starts from i_t and goes upwards in T' until the first marked ancestor $\mathrm{anc}(i_t)$ of i_t is reached. During this upward traversal, the algorithm marks each internal node of T' on the path connecting i_t to $\mathrm{anc}(i_t)$. Then, WTA starts from $\mathrm{anc}(i_t)$ and goes downwards in order to find the leaf $i' \in B$ closest to i_t. Note how the algorithm uses node marks for finding its way down: For instance, in Figure 1 the algorithm goes left since $\mathrm{anc}(i_t)$ was reached from below through the right child node, and then keeps right all the way down to i'. Node i'' (if present) is then identified via the links in B. The two distances $d(i_t, i')$ and $d(i_t, i'')$ are compared, and the closest node to i_t within B is then determined. Finally, WTA updates the links of B by inserting i_t between i' and i''.

In order to quantify the amortized time per trial, the key observation is that each internal node k of T' gets visited only twice during *upward* traversals over the n trials: The first visit takes place when k gets marked for the first time, the second visit of k occurs when a subsequent upward visit also marks the other (unmarked) child of k. Once both of k's children are marked, we are guaranteed that no further upward visits to k will be performed. Since the preprocessing operations take $\mathcal{O}(n)$, this shows that the total running time over the n trials is linear in n, as anticipated.[5]

[5]Note, however, that the worst-case time per trial is $\mathcal{O}(\log n)$. For instance, on the very first trial T' has to be traversed all the way up and down.

7. Experiments

We now present the results of an experimental comparison on a number of real-world weighted graphs from different domains: text categorization, optical character recognition, and bioinformatics.

Our goal is to compare the prediction accuracy of WTA to the one achieved by known baseline algorithms for weighted (and unweighted) graph prediction. We compare our algorithm to the following two other online prediction methods, intended as representatives of two different ways of facing the graph prediction problem: global vs. local prediction.

Perceptron with graph Laplacian kernel by Herbster & Pontil (2007), abbreviated as GPA (graph Perceptron algorithm). This predicts the nodes of a weighted graph $G = (V, E)$ after mapping V via the linear kernel based on $L_G^+ + \mathbf{1}\,\mathbf{1}^\top$, where L_G is the laplacian matrix of G. Following Herbster et al. (2009b), we run GPA on a spanning tree T of the original graph. We do so because computing the pseudoinverse L_G^+ when G is a tree takes time and space quadratic in the number of nodes n (this in contrast to WTA that runs in linear time and linear space). GPA is a global approach in the sense that the graph topology affects, via the inverse Laplacian, the prediction on all nodes.

Online Majority Vote (abbreviated as OMV). Since the common underlying assumption to graph prediction algorithms is that nearby nodes are labeled similarly, a very intuitive and fast algorithm for sequentially predicting the label of a node i_t is via a weighted majority vote on all labels of the adjacent nodes seen so far, i.e., $\mathrm{SGN}\big(\sum_{s<t\,:\,(i_s,i_t)\in E} y_{i_s} w_{i_s,i_t}\big)$. The overall time and space requirements are both of order $\Theta(|E|)$, since we need to read (at least once) the weights of all edges. OMV-like algorithms are local approaches, in the sense that prediction at one node is affected only by adjacent nodes. OMV, as presented above, is the most natural online version of the **label propagation** (or energy minimization) algorithm (Zhu et al., 2003), abbreviated as LABPROP, which we keep as an accuracy baseline throughout our experiments.[6] LABPROP is a batch transductive learning method and is computed by solving a (possibly sparse) linear system of equations which requires $\mathcal{O}(kn^2)$ time on an n-node graph with k neighbors per node. This bad scalability, which prevented us from carrying out comparative

[6]Many other algorithms have been proposed in the literature for graph prediction problems, including the label-consistent mincut approach of Blum & Chawla (2001) and a number of other "energy minimization" methods —e.g., the one in Belkin et al. (2004). See (Bengio et al., 2006) for a relatively recent survey on this subject.

experiments on larger graphs of 10^5 nodes, should be taken into account when comparing LABPROP to fast online (i.e., one-sweep) algorithms.

In our experiments, we combined WTA and GPA with spanning trees generated in different ways (note that OMV and LABPROP do not operate on spanning trees).

Random Spanning Tree (RST). Each spanning tree is taken with probability proportional to the product of its edge weights —see, e.g., Ch. 4 of (Lyons & Peres, 2009). In addition, we also tested WTA combined with RST generated ignoring the edge weights (which were restored before running WTA). As shown in (Wilson, 1996; Alon et al., 2008), it is possible to generate unweighted random spanning trees in time *linear* in the number n of nodes for many and important classes of graphs. This gives a prediction algorithm whose total running time (including the generation of the spanning tree) is $\mathcal{O}(n)$ for many graphs. We abbreviate this spanning tree as NWRST (non-weighted RST).

Depth-first spanning tree (DFST). The spanning tree is created via the following randomized depth-first visit: A root is selected at random, then each newly visited node is chosen with probability proportional to the weights of the edges connecting the current vertex with the adjacent nodes that have not been visited yet. This spanning tree is faster to generate than RST, and can be viewed as an approximate version of RST.

Minimum Spanning Tree (MST). The spanning tree minimizing the sum of the resistors of all edges. This is the tree whose Laplacian best approximates the Laplacian of G according to the trace norm criterion —see, e.g., (Herbster et al., 2009b).

Shortest Path Spanning Tree (SPST). Herbster et al. (2009b) use the shortest path tree for its small diameter (at most twice the diameter of G), which allows them to better control the theoretical performance of GPA. We generated n shortest path spanning trees by varying the choice of the root node, and then took the one having minimal diameter among them.

Finally, in order to check whether the information carried by the edge weight has predictive value for a nearest neighbor rule like WTA, we also performed a test by ignoring the edge weights during both the generation of the spanning tree and the running of WTA's nearest neighbor rule. This is essentially the algorithm analyzed in (Herbster et al., 2009a), and we denote it with NWWTA (non-weighted WTA). We combined NWWTA with (weighted) MST, that is the spanning tree on which WTA performs best.

We ran our experiments on four medium size real-world datasets: (1) The first 10,000 documents (in chronological order) of **RCV1**, with TF-IDF preprocessing and vector normalization; (2) the **USPS** dataset with features normalized into $[0, 2]$; (3) the dataset of Krogan et al. (2006); Pandey et al. (2007) abbreviated as **KROGAN**; (4) a second dataset (Pandey et al., 2007), abbreviated as **COMBINED**, resulting from a combination of three datasets from (Gavin et al., 2002; Ito et al., 2001; Uetz et al., 2000);

On the RCV1 and USPS datasets we generated graphs with as many nodes as the total number of examples $(\boldsymbol{x}_i, y_i)$, that is, 10,000 nodes for RCV1 and 7291+2007 = 9298 for USPS. Following previous experimental settings (Zhu et al., 2003; Belkin et al., 2004), we used k-NN based on the standard Euclidean distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ between node i and node j. The weight $w_{i,j}$ was set as $w_{i,j} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|/\sigma^2}$, if j is one of the k nearest neighbors of i, and 0 otherwise. To set σ^2, we first computed the average square distance between i and its k nearest neighbors, and then took a further average over i. On USPS we generated two graphs, **USPS-10** and **USPS-100**, by running k-NN with $k = 10$ and $k = 100$. On RCV1 we generated a single graph, **RCV1-100**, by setting $k = 100$. We selected the four most frequent categories in RCV1 and all 10 categories in USPS.

KROGAN and COMBINED are high-throughput protein-protein interaction networks of budding yeast taken from (Pandey et al., 2007). We only consider the biggest connected components of both datasets, obtaining 2,169 nodes and 6,102 edges for KROGAN, and 2,871 nodes and 6,407 edges for COMBINED. In these graphs, each node belongs to one or more classes, each class representing a protein function. We selected the set of functional labels at depth one in the *FunCat* classification scheme of the MIPS database (Ruepp, 2004), resulting in 17 classes per dataset.

In order to associate binary classification tasks with the five datasets/graphs (RCV1-100, USPS-10, USPS-100, KROGAN, and COMBINED) we binarized the corresponding multiclass problems via a standard one-vs-rest scheme. We thus obtained: 4 binary classification tasks for RCV1-100, 10 binary tasks for USPS-10 and USPS-100, 17 binary tasks for both KROGAN and COMBINED. For a given a binary task and dataset, we tried different proportions of training and test set sizes. On all datasets we used both the two-fold 50% train – 50% test and the four-fold 75% train – 25% test. The error rate results we report in Table 1 are obtained by (either two or four)-fold cross validation over the entire datasets after macro-averaging over the corresponding binary tasks.

In our experimental setup we tried to control the

Table 1. Macro-averaged and cross-validated classification error rates (percentages) achieved by the various algorithms on the five datasets/graphs mentioned in the main text. We compare WTA, GPA, and the algorithm by Herbster et al. (2009a) combined with different spanning trees, to LABPROP and OMV. TRAIN:TEST denotes the training and test set size ratio (for instance, 3:1 means 75% train and 25% test). In boldface are the lowest errors on each dataset/graph among the online algorithms (thus excluding LABPROP). Standard deviations (averaged over the binary problems) are quite small. For instance, in Krogan and Comb, the average standard deviations are below 1.0%.

DATASET	RCV1-100		USPS-10		USPS-100		KROGAN		COMBINED	
TRAIN:TEST	1:1	3:1	1:1	3:1	1:1	3:1	1:1	3:1	1:1	3:1
ALGORITHM										
WTA+RST	23.2	21.5	2.1	1.8	5.2	4.5	19.0	18.4	19.7	19.2
WTA+DF	20.4	18.7	2.0	1.6	4.2	3.5	18.7	18.1	19.7	19.1
WTA+MST	**11.8**	**10.2**	**1.0**	0.8	**1.0**	**0.9**	18.3	17.6	19.6	19.1
WTA+SPST	21.4	20.1	2.3	1.9	4.2	3.6	19.5	18.9	19.9	19.5
WTA+NWRST	23.8	22.0	2.4	2.0	5.8	5.0	19.4	18.7	19.9	19.5
GPA+RST	32.5	31.2	4.7	4.4	9.6	8.6	21.7	22.0	21.4	21.5
GPA+DF	41.2	40.1	24.0	18.7	28.8	23.6	24.1	22.6	23.8	22.7
GPA+MST	20.4	18.2	2.0	1.7	2.0	1.8	20.7	20.9	21.1	20.7
GPA+SPST	24.5	24.4	2.9	2.7	5.2	4.5	20.8	20.6	21.1	20.2
GPA+NWRST	32.1	31.4	6.0	5.4	10.1	9.9	21.8	21.5	22.0	22.0
NWWTA+NWDFST	21.4	20.3	2.5	2.3	5.2	4.8	19.3	19.0	19.9	19.7
NWWTA+MST	12.8	11.9	1.2	1.2	1.2	1.2	18.8	18.4	19.8	19.6
OMV	25.4	20.9	1.1	**0.7**	1.9	1.6	**16.3**	**16.0**	**17.5**	**17.3**
LABPROP	10.9	9.5	0.8	0.7	2.0	1.7	15.1	15.3	16.0	16.2

sources of variance as follows: (i) We first generated 10 random permutations of the node indices for each of the five graphs/datasets; (ii) on each permutation we generated the training/test splits, (iii) we computed MST and SPST for each graph and made (for WTA, GPA, OMV, and LABPROP) one run per permutation on each of the $4+10+10+17+17 = 58$ binary problems, averaging results over permutations and splits; (iv) we generated 10 RST's and 10 DFST's for each graph (possibly disregarding edge weights at either generation time or prediction time), and operated as in (ii), with a further averaging over the randomness in the tree generation.

Table 1 gives the average fraction of prediction mistakes achieved by the various algorithms on the five datasets/graphs. Though the experiments are not conclusive, several interesting observations can be made.

1. WTA outperforms GPA on all datasets and with all spanning tree combinations. In particular, though we only reported aggregated results, the same relative performance pattern among the two algorithms repeats systematically over all binary classification problems. In addition, WTA runs significantly faster than GPA, requires less memory storage (linear in n, rather than quadratic), and is also fairly easy to implement.
2. The best performing combination for both WTA and GPA is MST. This might be explained by the fact that MST tends to select light ϕ-edges of the original graph.
3. By comparing NWWTA to WTA, we see that the edge weight information in the nearest neighbor rule is beneficial.
4. On RCV1 and USPS the prediction performance of WTA+MST is comparable to that of LABPROP, whereas on KROGAN and COMBINED WTA+MST is slightly inferior. However, recall that LABPROP takes time $\mathcal{O}(kn^2)$, where k is the node degree, whereas a single sweep of WTA+MST over the graph just takes[7] time $\mathcal{O}(kn \log n)$.

   Moreover, a simple way of making WTA outperform LABPROP on the two biological datasets is to let WTA predict through a *committee* of RST's aggregated via a majority vote. For instance, using WTA with a committee of 11 RST's generated independently (either considering or disregarding the edge weights) gets the following figures.

DATASET	KROGAN		COMBINED	
TRAIN:TEST	1:1	3:1	1:1	3:1
ALGORITHM				
WTA+11RST	14.9	14.4	14.9	14.6
WTA+11NWRST	15.0	14.6	14.9	14.7
OMV	16.3	16.0	17.5	17.3
LABPROP	15.1	15.3	16.0	16.2

   Similar improvements are likely to occur on the other datasets. On USPS, WTA+MST, LABPROP, and OMV tend to perform comparably.
5. NWRST and DFST are fast approximations to RST. Though the use of NWRST and DFST does not provide the same theoretical performance guarantees as RST, in our experiments the three do actually

[7] The MST of a graph $G = (V, E)$ can be computed in time $\mathcal{O}(|E| \log |V|)$.

perform comparably. Hence, in practice, NWRST and DFST might be viewed as fast and practical ways to generate spanning trees for WTA.

8. Conclusions and ongoing research

We introduced and analyzed WTA, an online prediction algorithm for weighted graph prediction. The algorithm uses random spanning trees and has nearly optimal (expected) performance guarantees in terms of both prediction accuracy and running time. Our initial experimental evaluation shows that WTA outperforms other previously proposed online predictors. Moreover, when combined with an aggregation of random spanning trees, WTA also tends to beat standard batch predictors, such as label propagation. These features make WTA (and combinations thereof) suitable to large scale applications. We are currently performing a more thorough experimental investigation to confirm the above findings.

Acknowledgments

This work was supported in part by Google Inc. through a Google Research Award, and by the PASCAL2 Network of Excellence (EC grant no. 216886). This publication only reflects the authors views.

References

Alon, N., Avin, C., Koucký, M., Kozma, G., Lotker, Z., and Tuttle, M.R. Many random walks are faster than one. In *Proc. 20th Symp. on Parallel Algo. and Architectures*, pp. 119–128. Springer, 2008.

Belkin, M., Matveeva, I., and Niyogi, P. Regularization and semi-supervised learning on large graphs. In *Proc. 17th COLT*, pp. 624–638. Springer, 2004.

Bengio, Y., Delalleau, O., and Le Roux, N. Label propagation and quadratic criterion. In *Semi-Supervised Learning*, pp. 193–216. MIT Press, 2006.

Blum, A. and Chawla, S. Learning from labeled and unlabeled data using graph mincuts. In *Proc. 18th ICML*, pp. 19–26. Morgan Kaufmann, 2001.

Cesa-Bianchi, N., Gentile, C., and Vitale, F. Fast and optimal prediction of a labeled tree. In *Proc. 22nd COLT*. Omnipress, 2009.

Cesa-Bianchi, N., Gentile, C., Vitale, F., Zappella, G. Random spanning trees and the prediction of weighted graphs. Technical Report, Università degli Studi di Milano & Università dell'Insubria, 2010.

Chang, H. and Yeung, D.Y. Graph Laplacian kernels for object classification from a single example. In *2006 IEEE CVPR*, pp. 12011–2016. IEEE Press, 2006.

Gavin, A.-C. et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–147, 2002.

Goldberg, A. and Zhu, X. Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization. In *HLT-NAACL 2006 Workshop on Textgraphs: Graph-based algorithms for NLP*, 2004.

Herbster, M. and Pontil, M. Prediction on a graph with the Perceptron. In *NIPS 21*, pp. 577–584. MIT Press, 2007.

Herbster, M., Lever, G., and Pontil, M. Online prediction on large diameter graphs. In *NIPS 22*. MIT Press, 2009a.

Herbster, M., Pontil, M., and Rojas-Galeano, S. Fast prediction on a tree. In *NIPS 22*. MIT Press, 2009b.

Ito, T., Chiba, T., Ozawa, R., Yoshida, M., Hattori, M., and Sakaki, Y. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *PNAS*, 8(98):4569–4574, 2001.

Krogan, N.J. et al. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature*, 440:637–643, 2006.

Lyons, R. and Peres, Y. Probability on trees and networks. Manuscript, 2009.

Pandey, G., Steinbach, M., Gupta, R., Garg, T., and Kumar, V. Association analysis-based transformations for protein interaction networks: a function prediction case study. In *Proc. 13th ACM KDD*, pp. 540–549. ACM Press, 2007.

Ruepp, A. The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic Acids Research*, 32(18): 5539–5545, 2004.

Tsuda, H. Shin K. and Schölkopf, B. Protein functional class prediction with a combined graph. *Expert Systems with Applications*, 36:3284–3292, 2009.

Uetz, P. et al. Pa comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae. *Nature*, 6770(403):623–627, 2000.

Wilson, D.B. Generating random spanning trees more quickly than the cover time. In *Proc. 28th ACM STOC*, pp. 296–303. ACM Press, 1996.

Zhu, X., Ghahramani, Z., and Lafferty, J. Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.