Online Graph Prediction with Random Trees

Nicolò Cesa-Bianchi  
Università di Milano, Italy  
cesa-bianchi@dsi.unimi.it

Claudio Gentile  
Università dell’Insubria, Italy  
claudio.gentile@uninsubria.it

Fabio Vitale  
Università di Milano, Italy  
fabio.vitale@unimi.it

1 Introduction

In online transductive graph prediction a learner is initially given an undirected graph $G = (V,E)$ with unknown binary labels $y = (y_1, \ldots, y_n) \in \{-1, +1\}^n$ assigned to the vertices $V = \{1, \ldots, n\}$. At time $t = 1$ an arbitrary vertex $i_1 \in V$ is selected and the learner must predict its label $y_{i_1} \in \{-1, +1\}$. Then $y_{i_1}$ is revealed and a new vertex $i_2 \neq i_1$ of $V$ is selected. This process goes on for $t = 1, 2, \ldots, n$ until all vertices of $G$ have been selected. The learner’s goal is to minimize the number of prediction mistakes.

Online linear learners, such as the Perceptron algorithm, can be applied to this problem by embedding $V$ in $\mathbb{R}^n$ via a map that transforms node $i$ to the $i$-th coordinate versor $e_i \in \mathbb{R}^n$. For example, the graph Perceptron algorithm [4, 5] predicts the label of $e_i$ using the linear kernel $K = L_G^{-1} + 11^T$, where $L_G$ is the Laplacian of $G$, $L_G^{-1}$ its pseudoinverse, and $1 = (1, \ldots, 1)^T$. The resulting mistake bound is in general of the form $\Phi_G(y)(1 + R_G)$, where $\Phi_G(y)$ is the labelling cutsize, that is the number of edges in $G$ whose endpoints have different labels (we call these $\phi$-edges), and $R_G = \max_{i,j} r_{i,j}$ is the resistance diameter of $G$. Here $r_{i,j}$ denotes the effective resistance between $i$ and $j$.

A variant of this approach considers running the graph Perceptron on a spanning tree $T$ of $G$ [2]. The corresponding mistake bound is of the order $\Phi_T(y)D_T$, where $D_T$ is the diameter of the chosen tree. This improves on the cut size since $\Phi_T(y) \leq \Phi_G(y)$ but, on the other hand, $D_T = \Theta(D_G)$ where $D_G$ is the diameter of $G$. Note that $D_G$ can be much larger than $R_G$.

A different technique [3] attempts to control $D_T$ by linearizing $T$ via a depth-first visit. This gives a line graph $S$ (the so-called spine of $G$) such that $\Phi_S(y) \leq 2 \Phi_T(y)$. By running 1-Nearest Neighbor (1-NN) prediction on $S$, one can prove [3] the mistake bound $O(\Phi_T(y)\ln(n))$. In [2] it is suggested to pick $T$ in order to minimize the diameter $D_T$. However, since the adversary may concentrate all $\phi$-edges on the chosen tree $T$, there is no guarantee that $\Phi_T(y)$ remains small.

In this extended abstract two main issues are investigated: the best choice of $T$, and the best way of predicting once $T$ is given.

2 Prediction on a spanning tree

When labels $y$ on $V$ are adversarially chosen, the natural choice is to draw $T$ uniformly at random among all spanning trees of $G$. By exploiting Kirchoff’s equivalence between the effective resistance $r_{i,j}$ and the probability that $(i,j) \in E$ belongs to a random spanning tree, we immediately get the expected mistake bound $O(\Phi_R(y)\ln(n))$ for 1-NN on the linearized random tree, where $\Phi_R(y) = \frac{1}{2} \sum_{(i,j) \in E} r_{i,j}(y_i - y_j)^2$ is the resistance-weighted cutsize. This bound is significantly better than previous ones in certain cases. In fact, on an unweighted graph with $n$ nodes, the effective resistance $r_{i,j}$ of an edge $(i,j)$ always lies in
In particular, \( r_{i,j} \) is very small when \((i, j)\) is located in a densely connected area of the graph, while \( r_{i,j} = 1 \) when \((i, j)\) is a bridge edge. For instance, in a dense graph where \( r_{i,j} = \mathcal{O}(1/n) \) for all \((i, j)\) \(\in E\), the adversary may choose \( y \) so as to concentrate \( \Theta(n) \) \( \phi \)-edges on any specific tree, and yet \( \Phi_{R}(y) = \mathcal{O}(1) \). For “most” graphs, a random spanning tree can be sampled with a random walk in time \( \mathcal{O}(n \ln n) \) [1], although all known techniques take \( \Theta(n^{3}) \) in the worst case.

The next question we consider is whether one can further improve on this bound by efficiently predicting directly the nodes of \( T \), rather than applying 1-NN to the spine [3] or applying a suitable Laplacian-based kernel \( K \) to \( T \) [2]. To this end we introduce a new algorithm for predicting labelled trees.

The analysis of this algorithm is based on the simple notion of cluster. A cluster \( C \) of a labelled tree \( T \) is any maximal subtree of \( T \) containing no \( \phi \)-edges. We bound the number of mistakes made by the algorithm in terms of the cutsizes \( \Phi_{C}(y) \) and the diameter of the clusters which \( T \) is partitioned into by \( y \).

Let \( C \) be the set of all clusters of \( T \) induced by a given labelling \( y \), \( d_{C} \) be the diameter of cluster \( C \), and \( D_{C} = \max_{C \in C} d_{C} \leq D_{T} \) be the maximum diameter over all such clusters. Finally, let \( \Phi_{C}(y) \) be the number of \( \phi \)-edges incident to cluster \( C \).

**Theorem 1** [Proof omitted.] There exists an algorithm that, when run on a tree \( T \) with \( n \) nodes, makes a the number of mistakes bounded by

\[
\mathcal{O}\left(\sum_{C \in C} \Phi_{C}(y) \log d_{C}\right) = \mathcal{O}(\Phi_{T}(y) \log D_{C}).
\]

Both the total space required by the algorithm and the running time per label are \( \mathcal{O}(n) \).

Intuitively, this algorithm always tries to predict a node \( i_{t} \) with the label minimizing the size of the cut consistent with all labels seen so far. The algorithm does so by maintaining a collection of disjoint subtrees partially covering the graph. The label of the next node \( i_{t} \) is predicted according to whether the \( i_{t} \) belongs to a subtree or not. In the latter case, the algorithm predicts \( y_{i_{k}} \) through \( y_{k} \), where \( k \) is the node closest to \( y_{i_{t}} \) in the collection of subtrees. If the label of \( k \) is not known yet, then the algorithm acts as if it had been asked to predict \( y_{k} \) at time \( t \). In the special case when \( T \) is a line graph this algorithm is equivalent to 1-NN.

Details, performance analysis, and complexity analysis of the algorithm will appear in the full paper.

Since the above algorithm works for any tree, we could combine it with the random spanning tree approach mentioned at the beginning of this section. This would immediately yield a prediction bound holding in expectation for any undirected (and unweighted) graph. However, we believe the algorithm of Theorem 1 is interesting in its own right in that it efficiently deals with an important class of graphs.

We now briefly turn to lower bounds. The following is not hard to prove: given a line graph \( S \) with \( n \) nodes and any cutsize \( \Phi \leq n - 1 \), for any deterministic prediction algorithm there exists a labelling \( y \) such that the algorithm makes \( \Omega(\Phi \log(n/\Phi)) \) mistakes. This result can be used to build a more general lower bound that holds for any labelled tree. This lower bound has roughly the same form as the upper bound in Theorem 1, though it is expressed in terms of slightly different quantities.

**Theorem 2** Let \( T \) be a tree with \( n \) nodes and \( \Phi \) \( \phi \)-edges. Then an adversarial strategy exists that forces any deterministic algorithm to make at least order of \( \max_{1}^{Z-1} \phi_{i} \log(\ell_{i}/\phi_{i}) \) mistakes, where the maximum is over all possible decompositions of \( T \) into \( Z-1 \) line graphs with \( \ell_{1}, \ell_{2}, \ldots, \ell_{Z-1} \) edges, and over all possible allocations \( \phi_{1}, \ldots, \phi_{Z-1} \) of \( \Phi \) \( \phi \)-edges to them.

**Proof:** (Sketch) Any tree \( T \) with \( Z \) leaves can be always decomposed into \( Z-1 \) line graphs. One way of doing so is to operate in a sequence of \( Z-1 \) steps as follows. In the first step, let \( z_{1} \) and \( z_{2} \) be two leaf nodes.
of $T$ (say the ones furthest apart), and let $T_1$ be the line graph made up of the nodes on the (unique) path connecting $z_1$ and $z_2$ (including both ends). In the generic $i$-th step, a node $z_i$ not included in previously chosen paths is selected. Let $z'_i$ be the closest (w.r.t. the distance on the tree) node to $z_i$ among the nodes belonging to previously selected lines, and $T_i$ be the (unique) path connecting $z_i$ and $z'_i$ (including both $z_i$ and $z'_i$). Once this decomposition is performed (notice that many such decompositions are possible), we end up with a set of $Z - 1$ line graphs $T_i$ connected to one another by a single node. We clearly have $\sum_{i=1}^{Z-1} \ell_i = n - 1$, where $\ell_i$ in the number of edges of $T_i$. The adversary is then free to allocate a total of $\Phi \phi$-edges over the $Z - 1$ line graphs and then follow the above mentioned line graph adversarial strategy on each line independently. Denote by $\phi_i$ the number of $\phi$-edges allocated to $T_i$. A “legal” adversarial allocation satisfies $\phi_i \leq \ell_i$ and $\sum_{i=1}^{Z-1} \phi_i = \Phi$. \hfill \Box

We are currently working on the case when the nodes of the undirected graph (known ahead of time) have linear functions associated with them. A direct approach based on Laplacians kernels seem to lead to vacuous results.

References