Node Classification*

Ofir Geri, Or Ozery

School of Computer Science, Tel Aviv University, Israel.

Abstract. In this paper we review the Node Classification problem, in which the algorithm has to determine the labeling of samples (represented as nodes) by looking at the labels of their neighbors. This problem has many applications in social network studies and in recommendation systems. We review some of the basic algorithms for this problem in both offline and online scenarios. We also present a new randomized online algorithm and compare it to the other presented online algorithm.

1 Introduction

In a standard classification problem, we have a set of hypotheses $H$, from which we look for the “best” hypothesis $h_{\text{best}} \in H$, and use it to classify new samples. The classification of the sample $S = \{(x_i, y_i)\}_{i=1}^m$ is used to select $h_{\text{best}}$. Thus, the classification of a new point is implicitly affected by the classification of the sample points (since it affects $h_{\text{best}}$, which determines the classification of the new point).

In node classification, however, there is no set of hypotheses $H$. Instead, for each pair of samples we are given a correlation measure. Intuitively, we expect a sample which is strongly correlated to positive samples, to be positive. So unlike standard classification, here samples explicitly affect each other’s classification.

We can think of the samples as nodes in a graph, where edges represent the correlation measure between samples. We can use edge weights to indicate the correlation strength. Initially, only a subset of the nodes (samples) are classified - the nodes of the training set. The goal is to classify the entire graph, given the initial partial classification.

This problem is called Node Classification or Graph Labeling. It arises naturally in real-life problems that have an inherent network structure. The most obvious example is social networks, where one can evaluate a correlation between users that interact, and learn various features (such as age, personal preferences, etc.) about them. This information can be used, for example, in recommendation systems (such as friend suggestions, or simply focused advertisements). When the network structure is not given explicitly, we can try to build it using relevant information. For example, we can correlate between users who purchased the same items, or viewed the same content.

* Submitted as a final project in the course Machine Learning: Foundations, Fall 2012/13.
The rest of the paper is organized as follows: Section 2 formulates the node classification problem, and describes the general (offline) algorithm by Blum and Chawla [2]. Section 3 deals with the online variant of node classification, and compares two algorithms: the Majority algorithm by Brautbar [3], and our randomized algorithm. Section 4 concludes this paper.

2 The Node Classification Problem

2.1 Definition

The node classification problem deals with an undirected weighted graph \((V, E, w)\), and a labeling function \(f : V \rightarrow K, |K| = k\). The weight function \(w\) gives some indication about the possibility that neighbors will have the same label. It is still possible to have an edge with big weight between vertices of different labels, but we do expect that considering the entire graph, there will be few such cases.

The value of \(f\) is only known for a subset of vertices \(L \subseteq V\), called the labeled set. The problem objective in node classification is to extend the given labeling \(f : L \rightarrow K\) to the entire set of vertices \(f_{ALG} : V \rightarrow K\), by assigning a label to each vertex in the unlabeled set \(U = V \setminus L\). The goal is to minimize the number of mistakes, i.e. minimize \(|\{v \in U | f(v) \neq f_{ALG}(v)\}|\).

2.2 Algorithm

We will now describe an algorithm for binary \((k = 2)\) node classification, which is due to Blum and Chawla [2]. It uses a reduction to the minimum cut problem, which in turn can be solved in polynomial time using a maximum flow algorithm. The algorithm gives a classification which is “optimal”, in the sense that it achieves minimal total weight of edges connecting nodes of different labeling.

Algorithm

1. Add 2 nodes \(v_+, v_-\) to the graph.
2. Connect \(v_+\) to all vertices with a positive label, using edges of infinite weight.
3. Connect \(v_-\) to all vertices with a negative label, using edges of infinite weight.
4. Find a minimum \((v_+, v_-)\) cut (a cut that separates \(v_+\) and \(v_-\)) in the modified graph.
5. Label all nodes in the cut side of \(v_+\) as positive, and all nodes in the cut side of \(v_-\) as negative.

We note that the above algorithm can be naturally generalized to \(k > 2\), by adding vertices \(v_i\) for each label \(i \in K\). We would then have to solve a minimum \(k\)-cut problem, which can be solved in time \(O(|V|k^2)\) (see [4]).
3 The Online Variant of Graph Labeling

In this section, we address the online scenario of graph labeling. As in the offline case, the problem concerns an undirected weighted graph \((V, E, w)\) and a labeling function \(f : V \rightarrow K, |K| = k\). In each round:

- We are presented with an unlabeled node \(v\).
- We choose a labeling \(f_{ALG}(v) \in K\).
- We are presented with the true label of \(v\), \(f(v)\).

Our purpose is to minimize the number of errors incurred by the classification algorithm.

We note that the algorithms in this section will address the special case of binary classification \((k = 2)\), and we shall denote by 1 and \(-1\) the possible labeling of vertices.

3.1 The Majority Algorithm

A natural approach for solving the online binary graph labeling problem is taken in the Majority algorithm, which was suggested by Brautbar [3]. In this algorithm, the classification of a node is simply determined by the weighted majority of its neighbors’ labels.

**The Majority Algorithm**

*Input*: a partially-labeled weighted graph \(G\), and a previously unlabeled vertex \(v\).

1. Let \(S \subseteq V\) be the set of all previously labeled vertices that are neighbors of \(v\).
2. If \(|S| = 0\), predict ’1’.
3. Else, predict \(f_{ALG}(v) = \text{sign}(\sum_{u\in S} w(u,v)f(u))\).

We will now discuss the number of mistakes incurred by this algorithm, \(m_{majority}\).

We will first analyze the number of mistakes when the given graph is an unweighted clique, We will later show that for certain labeling functions, the Majority algorithm is optimal on such graphs.

**Theorem 1.** Let \(G = (V, E)\) be an unweighted clique, and let \(s\) be a sequence of vertices. Denote the number of vertices labeled with one labeling type by \(n_1\), where \(n_1 \leq n - n_1\). Then, \(m_{majority} \leq 2n_1 + 1\).

**Proof.** Assume without loss of generality that \(n_1\) is the number of vertices labeled ’1’. There are two types of mistakes: mistakes when classifying nodes that their true label is ’1’ and mistakes when classifying nodes that their true label is ’-1’. There are at most \(n_1\) mistakes of the former type since there are at most \(n_1\) nodes that their true label is ’1’. Assume we have made \(n_1 + 1\) mistakes of the latter type. Then each node that should be labeled ’-1’ will be labeled correctly,
since we’ve already labeled $n_1 + 1$ nodes that their label is ’−’, and the graph is a clique. Therefore, the total number of mistakes is at most $2n_1 + 1$. □

The upper bound proved above is useful in cases where the labeling function is unbalanced, i.e. one label is more common than the other. For unbalanced labeling functions where the common label is spread on more than $\frac{3}{4}$ of the vertices, this upper bound is also a lower bound for any deterministic algorithm. Thus, for such labeling functions, the majority algorithm is optimal. The lower bound is given in the following theorem.

**Theorem 2.** Let $G = (V, E)$ be an unweighted clique. For every deterministic algorithm $A$ there exists a labeling function $f$ that satisfies $n = 4n_1 + 1$ (where $n_1$ is defined as in Theorem 1) and a sequence of vertices $s$ that will make $A$ incur at least $2n_1 + 1$ mistakes.

*Proof.* We will construct the labeling $f$ adaptively based on $A$’s moves. We start by taking a set $S$ of $2n_1 + 1$ vertices, presenting each vertex to $A$, and then labeling it the opposite value. Thus $A$ will incur $2n_1 + 1$ mistakes on the first $2n_1 + 1$ vertices in the series. To complete the proof, we need to label the rest of the vertices in a way that will satisfy $n = 4n_1 + 1$.

We denote by $t$ the number of occurrences of the less common label so far. Since we only labeled $2n_1 + 1$ vertices so far, we know that $t \leq n_1$. We can therefore complete the labeling as follows: label the next $n_1 - t$ vertices with the less common label, and then label the rest of the vertices with the other label. □

We are now interested in bounding $m_{majority}$ for any weighted graph $(V, E, w)$. For arbitrary random labeling we cannot expect good results. But let us recall that we assumed that the labeling will tend to be similar for vertices connected by edges with big weights. In the offline case, we used a minimum cut to label the vertices. In the online case, we will use the cut size induced by a labeling as a measure of performance. Intuitively we expect the Majority algorithm to predict best when the cut size is small. We shall now define the cut size induced by a labeling $f$, and use it to bound $m_{majority}$.

**Definition 1.** The cut size induced by a labeling $f$ is defined as $\text{CutSize}(f) = |\{(u, v) \in E | f(u) \neq f(v)\}|$.

**Theorem 3.** For any graph $G$, true labeling $f$, and a sequence of prediction requests $s$, $m_{majority} \leq \text{CutSize}(f) + \alpha$ where $\alpha$ is the size of the maximum independent set in $G$.

*Proof.* We divide the vertices of $G$ into two groups:

$V_1$ - The set of vertices that by the time their label is predicted, none of their neighbors has already been labeled,

$V_2$ - The set of vertices that by the time their label is predicted, at least one of their neighbors has already been labeled.

We bound the number of mistakes in $V_1$ by $\alpha$, and the number of mistakes in $V_2$ by $\text{CutSize}(f)$. 
The size of $V_1$ is at most $\alpha$. We prove this by contradiction. Assume that there are more than $\alpha$ vertices in $V_1$. Then $V_1$ cannot be an independent set, and there's an edge connecting two of the vertices in $V_1$. By the time the second of them is labeled, its neighbor has already been labeled, so the node being labeled cannot be in $V_1$. We have reached a contradiction, hence $|V_1| \leq \alpha$, and the number of mistakes in $V_2$ is bounded by $\alpha$.

We now prove that the number of mistakes in $V_2$ is at most $\text{CutSize}(f)$. Every vertex in $V_2$ that is labeled incorrectly must have a neighbor that is already labeled with the false label. The number of edges connecting each incorrectly-labeled vertex with its neighbor is at most $\text{CutSize}(f)$. No edge is counted twice since one of the vertices connected by the edge is labeled first, that is before its neighbor has a label. Therefore, the number of mistakes in is bounded by $\text{CutSize}(f)$.

We conclude that $m_{\text{majority}} \leq \text{CutSize}(f) + \alpha$. \hfill \Box

We can improve the above bound, by counting vertices with conflicting neighbors, instead of counting conflicting edges. For this purpose we define the frontier size of a labeling $f$.

**Definition 2.** The frontier size of a labeling $f$ is defined as $\text{FrontierSize}(f) = |\{v \in V \mid \exists u \in V, (u, v) \in E, f(u) \neq f(v)\}|$.

**Theorem 4.** For any graph $G$, true labeling $f$, and a sequence of prediction requests $s$, $m_{\text{majority}} \leq \text{FrontierSize}(f) + \alpha$ where $\alpha$ is the size of the maximum independent set in $G$.

**Proof.** The proof is identical to the proof of Theorem 3, except that we bound the number of mistakes in $V_2$ by the frontier size instead of the cut size. Every vertex in $V_2$ that is labeled incorrectly must have a neighbor that is already labeled with the false label. Therefore, the incorrectly-labeled vertex belongs to the frontier. We get that the number of mistakes in $V_2$ is bounded by $\text{FrontierSize}(f)$. Therefore,$m_{\text{majority}} \leq \text{FrontierSize}(f) + \alpha$. \hfill \Box

### 3.2 Randomized Majority

We now introduce a randomized variant of the Majority algorithm. We assume the oblivious adversary model, in which the sequence of queries is chosen in advance (the adversary only knows the algorithm in use).

**The Randomized Majority Algorithm**

*Input:* a partially-labeled weighted graph $G$, and a previously unlabeled vertex $v$.

We assume that all weights are positive.

1. Let $S \subseteq V$ be the set of all previously labeled vertices that are neighbors of $v$, and $S^+ \subseteq S$ be the subset containing only vertices that are labeled ’1’.
2. If $|S| = 0$, predict ’1’ with probability 0.5 and ’−1’ with probability 0.5.
3. Else, predict ’1’ with probability 
\[ \frac{\sum_{u \in S^+} w(u, v)}{\sum_{u \in S} w(u, v)} \], and ’−1’ otherwise.

The algorithm is essentially the same as the Majority algorithm, except that we use the labels and weights of the neighbors of each vertex as a distribution, instead of deterministically choosing the more probable label.

**Lemma 1.** Let \( f \) be a labeling function of an unweighted clique graph with \( n_1 \leq n - n_1 \) vertices of one label. A sequence of \( 2n_1 \) vertices of alternating labels, followed by \( n - 2n_1 \) vertices of the common label will make the Randomized Majority algorithm have the maximum number of expected mistakes.

**Proof.** Let \( s = v_1, ..., v_n \) be a sequence of vertices. We want to prove that \( s \) isn’t worse than the sequence described in the lemma. If \( s \) is the same as described in the lemma, we have nothing to prove. Otherwise, the labels of the first \( 2n_1 \) vertices in the sequence are not strictly alternating. Let \( i \) be the minimal index such that \( f(v_i) = f(v_{i-1}) \). We know that \( i \leq 2n_1 \). Therefore, the other label (not the label of \( v_{i-1} \)) appeared so far at most \( \left\lfloor \frac{2n_1 - 1}{2n_1} \right\rfloor = n_1 - 1 \) times, and so it must appear at least one more time after \( i \). Let \( k > i \) be the index of that first time that the other label appears again. We will substitute \( v_i \) and \( v_k \) and call that sequence \( s' \). The sequence \( s' \) will make the Randomized Majority algorithm incur more mistakes than \( s \): After labeling \( v_k \), both sequences will have the same number of mistakes, and in \( s' \), all vertices between \( v_{i+1} \) and \( v_k \) are not labeled as the new \( v_i \), so the expected number of mistakes will increase. \( s' \) starts with a longer sequence of alternating labels, so we can keep using the same argument until we get to a sequence as described in the lemma, which will have the worst expected outcome. \( \square \)

**Theorem 5.** Let \( G = (V, E) \) be an unweighted clique, and let \( s \) be a sequence of vertices. Denote the number of vertices labeled with one labeling type by \( n_1 \), where \( n_1 \leq n - n_1 \). Then, the expected number of mistakes is 
\[
 n_1 \left( \ln \left( \frac{n-1}{2n_1-1} \right) + 1 \right) + \frac{1}{2} + \frac{1}{4} \ln (2n_1 - 1).
\]

**Proof.** Let’s analyze the expected number of mistakes on a sequence starting with vertices of alternating labels. This kind of sequence is proven in Lemma 1 to have the maximal number of expected mistakes. For every vertex, define an indicator variable that takes the value of 1 if the vertex is labeled incorrectly. The expected number of mistakes in the whole sequence is the sum of the probabilities of failing to label each node. We divide the sequence into two parts: the first \( 2n_1 \) vertices (having alternating labels) and the rest of the sequence. For the first part of the sequence, the expected number of mistakes is:

\[
\frac{1}{2} + \frac{1}{1} + \frac{1}{2} + \frac{2}{3} + \ldots + \frac{1}{2} + \frac{n_1}{2n_1 - 1} = \frac{1}{2} \cdot 2n_1 + \left( \frac{0.5}{1} + \frac{0.5}{3} + \ldots + \frac{0.5}{2n_1 - 1} \right) = n_1 + \frac{1}{2} + \frac{1}{2} \sum_{i=2}^{n_1} \frac{1}{2i - 1}
\]
\[
\leq n_1 + \frac{1}{2} + \frac{1}{2} \int_1^{n_1} \frac{dx}{2x - 1} = n_1 + \frac{1}{2} + \frac{1}{4} (\ln(2n_1 - 1) - \ln(1)) = n_1 + \frac{1}{2} + \frac{1}{4} \ln(2n_1 - 1)
\]

For the rest of the sequence, the expected number of mistakes is:

\[
\frac{1}{2} + \frac{n_1}{2n_1 + 1} + \ldots + \frac{n_1}{n - 1} = \sum_{i=2n_1}^{n-1} \frac{n_1}{i} = n_1 \cdot \sum_{i=2n_1}^{n-1} \frac{1}{i} \leq n_1 \cdot (\ln(n - 1) - \ln(2n_1 - 1)) = n_1 \cdot \ln\left(\frac{n - 1}{2n_1 - 1}\right)
\]

Combining these two results, we get that the total number of expected mistakes is \(n_1 (\ln(\frac{n-1}{2n_1 - 1}) + 1) + \frac{1}{2} + \frac{1}{4} \ln(2n_1 - 1)\). \hfill \square

Comparing the results of theorem 1 and theorem 5, we see that Randomized Majority achieves better results when:

\[
n_1 (\ln(\frac{n-1}{2n_1 - 1}) + 1) + \frac{1}{2} + \frac{1}{4} \ln(2n_1 - 1) \leq 2n_1 + 1.
\]

This happens (for large \(n\)) when \(n_1 \geq \frac{n}{2e} \approx 0.184n\). In particular, our randomized algorithm suppresses the deterministic lower bound given in Theorem 2. This lower bound served us as a motivation for developing a randomized online algorithm.

We note that regarding general weighted graphs, the Randomized Majority algorithm achieves the same bounds that we proved for the deterministic Majority algorithm. The proofs are identical.

4 Conclusion

We reviewed the node classification problem, a research area in machine learning that has many important application, and is also unique in its perspective, which uses graph theory and combinatorial optimization. The algorithms we presented are all based on graph minimum cuts. We note that there are node classification algorithms which take different, less graph-oriented approaches. They are further discussed in [1].

In this paper we also presented a new algorithm, Randomized Majority, which is better in some cases than its deterministic counterpart. In some cases it also better than any other deterministic algorithm for the online node classification problem.
References

2. A. Blum, S. Chawla. Learning from Labeled and Unlabeled Data using Graph Min-cuts. *ICML 01*.
3. M. Brautbar. Online Learning a Binary Labeling of a Graph. *MLG 09*.