A Scalable Multiclass Algorithm for Node Classification

Giovanni Zappella GIOVANNI.ZAPPELLA@UNIMI.IT
Università degli Studi di Milano, via Saldini 50, Milan, Italy

Abstract

We introduce a scalable algorithm, MUCCA for multiclass node classification in weighted graphs. Unlike previously proposed methods for the same task, MUCCA works in time linear in the number of nodes. Our approach is based on a game-theoretic formulation of the problem in which the test labels are expressed as a Nash Equilibrium of a certain game. However, in order to achieve scalability, we find the equilibrium on a spanning tree of the original graph. Experiments on real-world data reveal that MUCCA is much faster than its competitors while achieving a similar predictive performance.

1. Introduction

Classification of networked data is a quite attractive field with applications in computer vision, bioinformatics, spam detection and text categorization. In recent years networked data have become widespread due to the increasing importance of social networks and other web-related applications. This growing interest is pushing researchers to find scalable algorithms for important practical applications of these problems. In this paper we focus our attention on a task called *node classification*, studied in the semi-supervised setting Zhu et al. (2003). Recent work has studied the problem of finding scalable algorithms from a theoretic point of view with interesting results. For example Cesa-Bianchi et al. (2009; 2010; 2011) developed on-line fast predictors for weighted and unweighted graphs and Herbster et al. created a version of the Perceptron algorithm to classify the nodes of a graph (Herbster et al. (2009)). Other interesting papers on the topic are Kleinberg & Tardos (1999) and Herbster & Pontil (2006).

Appearing in the proceedings of the *Mining and Learning Workshop* (2012).

Recently, Erdem & Pelillo (2011) introduced a game-theoretic framework for node classification. We adopt the same approach and, in particular, we obtain a scalable algorithm by finding a Nash Equilibrium on a special instance of their game. The main difference between our algorithm and theirs is the high scalability achieved by our approach. This is really important in practice, since it makes possible to use our algorithm on large scale problems.

We have to report that while we were working on this paper a similar algorithm with completely different motivations has been independently presented by Zhang et al. (2011).

2. Basic Framework

Given a weighted graph $G = (V, E, W)$, a labeling of G is an assignment $y = (y_1, .., y_n) \in \{0, 1, ..., c\}^n$ where $n = |V|$.
We expect our graph to respect a notion of regularity where adjacent nodes often have the same label: this notion of regularity is called *homophily*. Most machine learning algorithms for node classification (Herbster et al. (2009); Cesa-Bianchi et al. (2010); Zhu et al. (2003); Cesa-Bianchi et al. (2011)) adopt this bias and exploit it to improve their performances.
The learner is given the graph G, but just a subset of y, that we call training set. The learner's goal is to predict the remaining labels minimizing the number of mistakes.

Cesa-Bianchi et al. (2010) also introduce an irregularity measure of the graph G, for the labeling y, defined as the ratio between the sum of the weights of the edges between nodes with different labels and the sum of all the weights. Intuitively, we can view the weight of an edge as a similarity measure between two nodes. We expect highly similar nodes to have the same label, and edges between nodes with different labels being "light". Based on this intuition, we may assign labels to non-training nodes to minimize some function of the induced weighted cut.

In the binary classification case, algorithms based on min-cut have been proposed in the past (for example Blum & Chawla (2001)). Generalizing this approach to the multiclass case, naturally takes us to the *multi-way cut* (or multi-terminal cut — see Costa et al. (2005)) problem. Given a graph and a list of terminal nodes, find a set of edges such that, once removed, each terminal belongs to a different component. The goal is to minimize the sum of the weights of the removed edges.

Unfortunately, the multi-way cut problem is MAX SNP-hard when the number of terminals is bigger than two (Dalhaus et al. (1994)). Furthermore, we should keep in mind two more points: 1. in some practical applications entities in the network will behave rationally (i.e. when people choose a costly service), 2. the approximation of min-cut using a smooth function may not be good enough when the training set is small.

3. Graph Transduction Game

In this section we describe the game introduced by Erdem & Pelillo (2011) that, in a certain sense, aims at distributing the cost of approximating the multi-way cut over the nodes. This is done by expressing the labels assignment as a Nash Equilibrium. We have to keep in mind that, since this game is non-cooperative, each player maximizes its own payoff disregarding what it can do to maximize the sum of utilities of all the players (the so-called social welfare). The value of the multi-way cut is strongly related to the value of the social welfare of the game, but in the general case a Nash Equilibrium does not give any guarantee about the collective result.

In the Graph Transduction Game (later called GTG), the graph topology is known in advance and we consider each node as a player. Each possible label of the nodes is a pure strategy of the players. Since we are working in a batch setting, we will have a train/test split that induces two different kind of players: **determined players**(I_D) those are nodes with a known label (train set), so in our game they will be players with a fixed strategy (they do not change their strategy since we can not change the labels given as training set) and **undetermined players**(I_U) those that do not have a fixed strategy and can choose whatever strategy they prefer (we have to predict their labels).

The game is defined as $\Gamma = (I, S, \pi)$, where $I = \{1, 2, ..., n\}$ is the set of players, $S = \times_{i \in I} S_i$ is the joint strategy space (the Cartesian product of all strategy sets $S_i \subseteq \{1, 2, ...c\}$), and $\pi : S \to \mathbb{R}^n$ is the combined payoff function which assigns a real valued payoff $\pi_i(s) \in \mathbb{R}$ to each pure strategy profile $s \in S$ and player $i \in I$.

A mixed strategy of player $i \in I$ is a probability distribution x over the set of the pure strategies of i. Each pure strategy k corresponds to a mixed strategy where all the strategies but the k-th one have probability equals to zero.

We define the utility function of the player i as

$$u_i(s) = \sum_{s \in S} x(s) \pi_i(s)$$

where $x(s)$ is the probability of s.

We assume the payoff associated to each player is additively separable (this will be clear in the following lines). This makes GTG a member of a subclass of the multi-player games called polymatrix games. For a pure strategy profile $s = (s_1, s_2, ...s_n) \in S$, the payoff function of every player $i \in I$ is:

$$\pi_i(s) = \sum_{j \sim i} w_{ij} \mathbb{I}_{\{s_i = s_j\}}$$

where $i \sim j$ means that i and j are neighbors, this can be written in matrix form as

$$\pi_i(s) = \sum_{j \sim i} A_{ij}(s_i, s_j)$$

where $A_{ij} \in \mathbb{R}^{c \times c}$ is the partial payoff matrix between i and j, defined as $A_{ij} = I_c \times w_{ij}$, where I_c is the identity matrix of size c and $A_{ij}(x, y)$ represent the element of A_{ij} at row x and column y. The utility function of each player $i \in I_U$ can be re-written as follows:

$$\begin{aligned} u_i(s) &= \textstyle\sum_{i \sim j} x_i^T A_{ij} x_j \\ &= \textstyle\sum_{i \sim j} w_{ij} x_i^T x_j \\ &= \textstyle\sum_{i \sim j} w_{ij} \sum_{k=1}^{c} x_{i_k} x_{j_k} \end{aligned}$$

where k is an action selected from the player's set and in case i is a determined node with training label k, x's components will be always zeros except the k-th corresponding to the pure strategy k. Since the utility function of each player is linear, it is easy to see that players can achieve their maximum payoff using pure strategies.

In a non-cooperative game, a vector of strategies S_{NE} is said to be a (pure strategies) Nash Equilibrium, if $\forall i \in I$, $\forall s'_i \in S_i : s'_i \neq s_i \in S_{NE}$, we have that

$$u_i(s_i, S_{NE}^{-i}) \geq u_i(s'_i, S_{NE}^{-i})$$

where $u_i(s_i, S^{-i})$ is the strategy configuration S except the i-th one, that is replaced by s_i. In practice, no

player i will change its strategy s_i to an alternative strategy improving its payoff.
There are no guarantees that the Nash Equilibrium exists in pure strategies, but *any game with a finite set of players and finite set of strategies has a Nash Equilibrium in mixed strategies* (Nash (1951), also see Nisan et al. (2008)). In this case each player does not have to choose a strategy but it mixes its choices over its strategies. Instead of maximizing its payoff, it will maximize its expected payoff.

With a slight abuse of terminology, we hereafter refer to labels or pure strategies with the same meaning.

3.1. The Evolutionary Stable Strategies approach

Erdem & Pelillo (2011) propose to find a Nash equilibrium of the GTG using the Evolutionary Stable Strategies. We briefly present their approach for the reader to better understand the difference between their algorithm and ours.

The evolutionary stable strategies (ESS) approach is well known (Weibull (1995)) in the game-theoretic literature. It considers a game played repeatedly; each repetition of the game is seen as a generation, where an imaginary population evolves through a selection mechanism that, at each step, gives to the best "choices" a growing portion of the total population.

The algorithm (later called GTG-ESS), at each generation, updates the probability associated to every action h of every player i as

$$x_{ih}(t+1) = x_{ih}(t)\frac{u_i(e_h)}{u_i(x(t))}$$

The previous formula is just the discrete version of the so-called multi-population replicator dynamic:

$$\dot{x}_{ih} = x_{ih}(u_i(e_h, x_{-i}) - u_i(x))$$

where e_h is a vector of zeros except the h-th component that is one and x_{ih} is the h-th strategy of player i. The fixed points of the previous equations are Nash Equilibria, and the discrete version has the same properties — for further details see Erdem & Pelillo (2011).

In this case, the computational cost of finding the Nash Equilibrium is $O(k|V|^2)$ where k is the number of iterations and considering the number of classes as a constant factor. Erdem & Pelillo (2011) experimentally found that the number of iterations grows linearly with the number of nodes, so they consider the running time close to $O(|V|^3)$, but they do not seem to have any upper bound on the number of iterations.

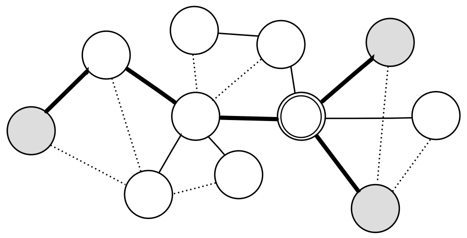


Figure 1. Some of the elements introduced in this paragraph: grey nodes are labeled nodes, the node with two circles is a fork, the fat black edges are those flagged "black line", the thin solid black line edges are edges of the spanning tree and the dotted edges are edges of the original graph not selected for the spanning tree.

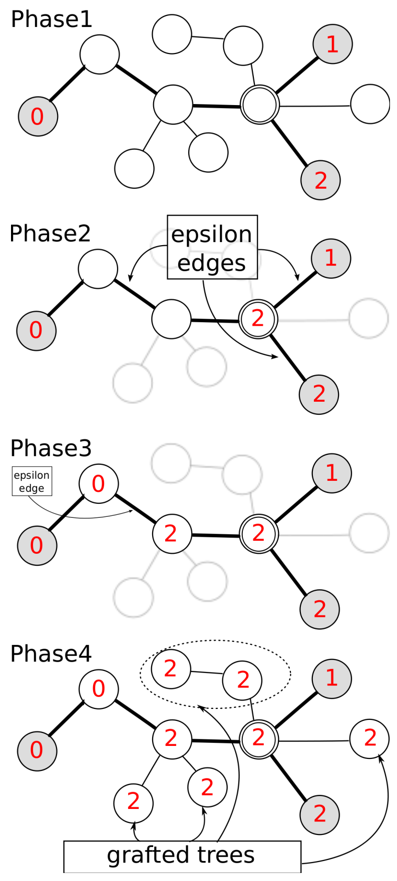
4. The *MUCCA* algorithm

Use GTG-ESS for large scale networks can not be considered a viable solution, even if the time complexity of the algorithm were to be demonstrated in the order of $\Theta(|V|^3)$. A possible alternative is to apply some known results about regret minimization, such as those described by Cesa-Bianchi & Lugosi (2006), in order to converge to a weaker notion of equilibrium, for example the Correlated Equilibrium (Aumann (1974)). Unfortunately, the results of our experiments with the Correlated Equilibrium were not satisfactory.

In this section we present *MUCCA*: a multiclass Classification Algorithm. Our algorithm consists in finding a Nash Equilibrium of the Graph Transduction Game on a special graph: a tree. We will show that in this way we achieve both good accuracy and scalability. In the remaining part of this section we will assume that the graph G is a tree.

Now, we briefly introduce few notions those will be useful later in this section (see Figure 1):

- **Revealed node**: a node whose label is known
- **Unrevealed node**: a node whose label is unknown
- **Fork**: an unrevealed node that is connected to at least 3 revealed nodes by edge-disjoint paths
- **Hinge node**: a revealed node or a fork
- **Hinge tree**: component of the forest created by removing from G all the edges incident to hinge nodes
- **Native hinge tree**: component of the forest created by removing from G all the edges incident to revealed nodes. Its connection nodes are intended to be only revealed nodes.
- **Hinge line**: a path connecting two hinge nodes



such that there are no internal nodes those are hinge nodes

- **Connection Node**: an hinge node that is connected to a node in an hinge tree
- **ϵ-edge**: given $\mathcal{P}_{ij}$, the path between i and j, an ϵ-edge is $\epsilon_{ij} \in \arg\min_{e \in \mathcal{P}_{ij}} w_e$, where w_e is the weight of the edge e.
- **Grafted tree**: a tree without hinge nodes that is connected to just one node on an hinge line

MUCCA works in four phases:

1. Mark all the paths between revealed nodes and find all the fork nodes
2. Estimate the label of each fork node
3. Assign a label to all the nodes on the hinge lines using a min cut technique
4. Assign a label to all the remaining nodes

Now we describe each phase of *MUCCA* in detail:

Phase 1

Starting from each revealed node, *MUCCA* does a breadth-first search until another revealed node or a leaf is found. Then if a revealed node was found, during the backtracking process, *MUCCA* marks the edges on the path to the starting node with a special flag that we will call “black line”. After that, each node with more than 3 edges marked as “black line” incident to it, is a fork.

Phase 2

Given a native hinge tree $\mathcal{H}$ that contains the fork $\mathcal{F}$, we can categorize its connection nodes into c categories using their labels. For each path between $\mathcal{F}$ and each connection node of $\mathcal{H}$, we have an ϵ-edge as defined before. The label assigned to $\mathcal{F}$ is the same as the category (of the connection nodes) that has the maximum sum of weights over the distinct ϵ-edges on the path between $\mathcal{F}$ and the nodes of that category.

Phase 3

On every hinge line, we label the nodes using min cut: in case the hinge nodes at the beginning and at the end of the line has the same label, all the nodes on the hinge line will be labeled with that label. Otherwise, all the nodes before the ϵ-edge are assigned with the label of the node at the beginning of the line, and the others with the label of the node at the end of the line. In case we have more than one edge with the same weight of the ϵ-edge (for example all the edges have the same weight), we use nearest neighbor to find the closest revealed node in order to complete the labeling of nodes in the line.

Phase 4

All the remaining nodes are in grafted trees. In this case, we assign the label of the node on the hinge line (connected to the tree) to all the nodes in that particular grafted tree.

We now prove that *MUCCA* finds a Nash Equilibrium for this special case of the GTG.

Theorem. *The labeling found by MUCCA is a Nash Equilibrium of the Graph Transduction Game when the graph is an undirected tree.*

Proof. As we explained in Section 3, a profile of strategies S_{NE} is a Nash Equilibrium if no one has incentive to deviate from its strategy. This means that $\forall i \in I,\ u_i(s_i, S_{NE}^{-i}) \geq u_i(s'_i, S_{NE}^{-i})$. For the purpose of contradiction suppose there exists a node j such that it can improve its payoff by changing its strategy.

j can not be contained within a grafted tree (those labeled in phase 4) since all the nodes contained in those trees have the same labels, so, whatever is the label, each of them gets a payoff of $\sum_{i \sim j} w_{ij}$, the maximum possible payoff.

j can not be on an hinge line since they are labeled using min cut, so in the best case the payoff of each node is already the maximum payoff; in the worst case the payoff is the maximum minus the weight of the ϵ-edge. Since the ϵ-edge, by definition, has the minimum weight, there is no chance to improve the payoff.
j can not be a revealed node (for obvious reasons).
j can not be a fork. Since we use min cut to label the hinge lines if the ϵ-edge of an hinge line is not incident to the fork, the node adjacent to the fork on that hinge line will have the same label of the fork. In this way the fork will get the part of payoff given by the edge between it and the adjacent node. Even if the ϵ-edges are incident to the fork, the label prediction can not achieve a payoff better than the one achieved by the majority vote.
Since j can not be a revealed node, nor a fork, nor a node on an hinge line, nor a node on a tree with just a connection node, it can not be in G. □

The implementation of *MUCCA* presented in this paper runs in $O(f|V|)$ where f is the number of forks, but it is possible to write a better implementation that runs in $O(|V|)$. Anyway this implementation of *MUCCA* can be already used in practice since the number of forks is usually really low.

Note that the labels of the unlabeled nodes of every native hinge tree can be predicted using only information about that singular native hinge tree. In this way, once the tree is splitted into native hinge trees, the predictions for the labels contained in every subtree are independent from the other subtrees. So, they can be computed using different threads, processes or even machines and we just need to get back a list of points (node id, predicted label). We would like to point out that *MUCCA* is scalable not because it if parallelizable, but because it scales linearly with time. it is possible to parallelize a consistent part of the algorithm.

5. Experiments

In this section we present our experimental results. Since binary classification is a (well-studied) special case of our problem, we first compare *MUCCA* with the state of the art in this particular field. In the second part of this section we test our algorithm against its competitors on multiclass problems. We have to keep in mind that *MUCCA* works in linear time and can be used on large-scale graphs, where we can not test its competitors.

Our **experimental protocol** is quite simple: for every size of the training set (and possibly on every class), we did 10 runs for each algorithm. Algorithms working on trees were ran on: Maximum Spanning Trees (or Minimum Spanning Tree if you consider the resistance distance on the tree as in Cesa-Bianchi et al. (2011)) since previous experimental works showed that predictors get their best results on this kind of tree (Cesa-Bianchi et al. (2011)), and on Random Spanning Trees (RandomSpanningTrees) generated as in Wilson (1996) to test the most scalable solution possible. We tested *MUCCA* also in a committee version: in the tables numbers before the predictor's name represent the number of predictors in the committee (for example n*MyPred represents a committee of n predictors MyPred). Each predictor in the committee predicts its own labels using its own tree and then we aggregate the predictions with a majority vote over the committee.

In order to better understand the competitiveness of *MUCCA*, we will compare it with some well-know **algorithms** besides GTG-ESS.

Label Propagation (abbreviated LABPROP), introduced by Zhu et al. (2003), is a popular algorithm for node classification and one of the most accurate algorithms in the literature. LABPROP computes a real-valued function $f : V \to \mathbb{R}$ on G, and then assigns a labeling to G using the values of f. The algorithm minimizes the following quadratic energy function:

$$E(f) = \frac{1}{2} \sum_{i \sim j} w_{ij} (f(i) - f(j))^2$$

with constrains on the labeled nodes. Its running time is $O(|E| \times |V|)$.

SHAZOO is a quite new algorithm introduced by Cesa-Bianchi et al. (2011) and, at the best of our knowledge, it is the most accurate scalable algorithms among our competitors. Also SHAZOO works on trees, but it mixes mincut and nearest neighbor. In

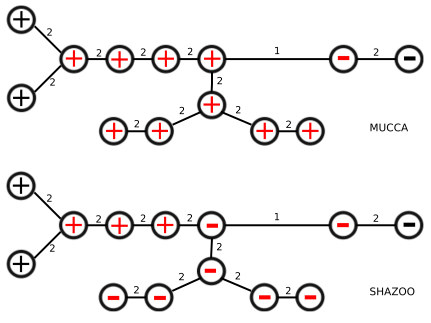


Figure 3. Red labels are predicted respectively by *MUCCA* in the first tree and SHAZOO in the second one

	RCV1				USPS			
	0.5%	1%	2%	5%	0.5%	1%	2%	5%
$MUCCA$+MinimumSpanningTree	27.70	25.34	23.12	18.85	3.24	2.06	1.35	1.10
$MUCCA$+RandomSpanningTree	32.02	29.41	27.06	22.83	9.20	7.49	5.78	4.26
SHAZOO+MinimumSpanningTree	26.87	24.35	21.64	18.26	2.92	1.85	1.29	1.08
SHAZOO+RandomSpanningTree	30.77	28.16	25.70	21.99	8.72	7.26	5.49	4.17
3*$MUCCA$+RandomSpanningTree	28.30	25.25	22.95	18.75	6.65	5.29	3.80	2.46
3*SHAZOO+RandomSpanningTree	27.29	24.44	22.24	18.44	6.67	5.52	3.84	2.48
7*$MUCCA$+RandomSpanningTree	26.09	23.17	20.62	16.61	6.19	4.60	3.14	1.78
7*SHAZOO+RandomSpanningTree	25.71	22.80	20.55	16.61	6.50	4.97	3.32	1.86
11*$MUCCA$+RandomSpanningTree	25.39	22.40	19.92	15.97	6.12	4.43	2.90	1.59
11*SHAZOO+RandomSpanningTree	25.30	22.37	20.01	16.09	6.46	4.85	3.15	1.69
GTG-ESS	21.66	19.13	17.00	14.37	2.69	1.67	1.17	0.92
LABPROP	26.74	23.51	20.84	16.37	7.34	5.27	3.65	2.36

Table 1. RCV1 and USPS - Error rate of each algorithm, the percentages in the first row refers to the size of the training set. Results are averaged over the 4 and 10 binary problems.

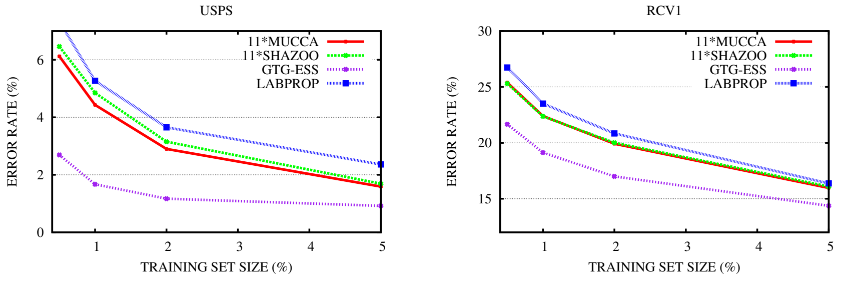


Figure 2. Macro-averaged test error rates on the binarized datasets as a function of the training set size. We plotted just the fastest combinations for SHAZOO and $MUCCA$, the one using Random Spanning Trees, but sometimes slightly better performances can be achieved using the Minimum Spanning Tree.

Figure 2, we see a toy example where SHAZOO and $MUCCA$ behave in a different way: $MUCCA$ predicts all the nodes on the left hand side as positive since there are two nodes labeled as positive, connected to a fork that is labeled as positive and the ϵ-edge is the one with weight 1. Clearly every player wants to maximize its payoff, so the algorithm produces a labeling with a minimum possible cut. In the second part, for SHAZOO the node in the middle looks closer to the negative example than to the positive ones. In this way also all the nodes contained into the grafted tree will be labeled in a different way.
SHAZOO's running time is $O(|V|)$.
Weighted Majority Vote (later called WMV) predicts the label of a node i using a majority vote on the labeled neighbors weighted on the edges connecting them to i. Its running time is $\Theta(|E|)$.

Both LABPROP and GTG-ESS can be used for multiclass datasets but it is not clear if it is possible to modify SHAZOO in order to get a multiclass algorithm. In the multiclass section we replace SHAZOO with WMV. We did not include in our comparison WTA (Cesa-Bianchi et al. (2010)) since it was always outperformed by SHAZOO. Graph Perceptron (Herbster et al. (2009)) was omitted since it requires a lot of computational resource and performed poorly in previous comparisons (for example in Cesa-Bianchi et al. (2010)).

5.1. Binary classification

In order to create a fair comparison, for the experimental activity we will generate our datasets in the same way of Cesa-Bianchi et al. (2011).

5.1.1. Datasets

We choose two real-world and well-known datasets to generate our graphs: USPS and RCV1.
USPS is a set of hand written digits collected by the United States Postal Service, it contains 9298 images of 16×16 pixels (gray scale); the dataset called **RCV1** is a subset of 10000 articles in chronological order taken from Reuters Corpus (a huge collection of news released by). All the articles have been pre-processed using TF-IDF. Both datasets were natively multiclass, so we tested our binary predictors using a standard one-vs-rest schema. We have 10 binary experiments for USPS and 4 binary experiments for RCV1.

We generated our graphs with as many nodes as the total number of examples for each dataset, keeping for each node only 10 nearest neighbor (before symmetrization) using the Euclidean distance $\|x_i - x_j\|$. Edges' weights have been calculated as

$$w_{ij} = e^{-\|x_i - x_j\|/\sigma^2}$$

where σ is the average between the weights of all the edges incident to i or j.

	USPS				CARDIO				GHGRAPH			
	0.5%	1%	2%	5%	0.5%	1%	2%	5%	0.5%	1%	2%	5%
$MUCCA$+MinimumSpanningTree	19.14	8.5	6.84	5.49	31.32	28.96	26.94	27.58	-	-	-	-
$MUCCA$+RandomSpanningTree	48.92	39.84	30.54	20.6	34.88	32.54	30.01	29.66	65.70	62.01	57.68	51.37
7*$MUCCA$+RandomSpanningTree	31.79	18.67	12.23	7.87	34.44	28.03	25.13	24.32	55.33	51.14	47.56	43.55
11*$MUCCA$+RandomSpanningTree	29.56	14.89	10.06	6.8	33.99	26.82	24.86	23.98	51.88	48.64	46.25	42.92
WMV	84.28	78.96	69.28	47.48	64.47	62.35	57.05	47.08	74.03	67.50	60.15	50.37
LABPROP	43.73	16.22	9.01	5.82	28.24	24.42	23.32	22.21	-	-	-	-
GTG-ESS	14.37	7.75	5.28	4.6	40.65	36.91	28.81	26.29	-	-	-	-

Table 2. Multiclass experiments - Error rate of each algorithm, the percentages in the first row refers to the size of the training set. Please note that we did not run GTG-ESS and LABPROP on GHGRAPH due to limited computational resources.

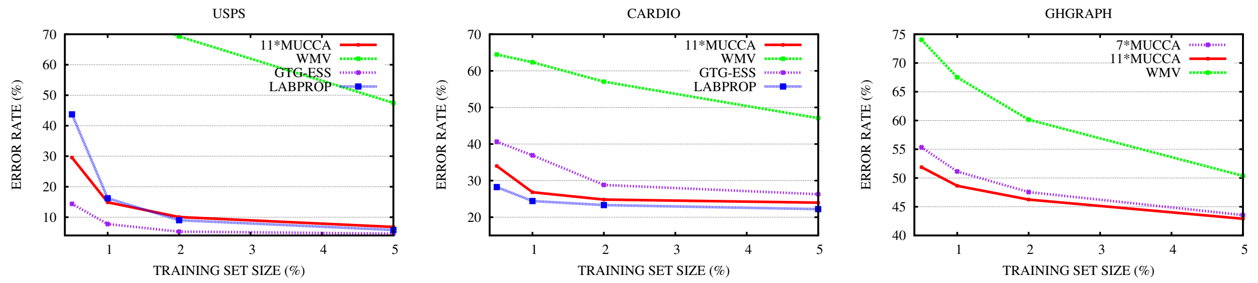


Figure 4. Test error rates on the multiclass datasets as a function of the training set size. Note that we plotted $MUCCA$'s result with Random Spanning Tree: the most scalable solution, but on USPS $MUCCA$ can achieve even better results with the Minimum Spanning Tree that can be generated in $O(|E| \log |V|)$.

5.1.2. Experimental results

The results, shown in Table 1, are not conclusive, but we can observe some interesting trends:
- GTG-ESS is the most accurate algorithm, but its computational complexity makes it not particularly suitable for most of the current practical applications.
- $MUCCA$'s accuracy is good and it is always close or better that SHAZOO's one.
- LABPROP approaches the competitors as the training set size grows.

5.2. Multiclass classification

In the multiclass comparison we replaced RCV1 with two other datasets, since it is not just multiclass but also multi-label (an element has more than one label). As we explained before, SHAZOO works just on binary problems, so the only scalable competitor in this section is WMV.

5.2.1. Datasets

USPS is the same graph we used for the binary classification.
CARDIO *consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians.* We generated a graph starting from the feature vectors, in the same way we did for USPS and RCV1. The graph is constituted by 2126 nodes and 13696 edges. The nodes are divided in 3 classes (fetal states)[1].
GHGRAPH is a graph created from the data released by Github.com for their contest in 2009. Every node represents a repository and an edge means that there is a developer working on both repositories connected by that edge. In case more than one developer works on those repositories, the number of developers is used as a weight for that edge. The label of each repository is the most used programming language to write the code in that repository. We kept only the biggest connected component of the graph that includes 99907 nodes (64885 of them are labeled) and 11044757 edges. Each programming language is a different class and at the end we had 40 classes.

5.2.2. Experimental results

The results of our experiments, shown in Table 2, are not conclusive, but we can observe some interesting trends: - It is not really clear which one between GTG-ESS and LABPROP is the most accurate algorithm, but anyway $MUCCA$ is always competitive with them.
- $MUCCA$ is always much better than WMV. As expected WMV works better on "not too sparse" graphs such GHGRAPH, but even in this case it is outperformed by $MUCCA$.
- GTG-ESS and LABPROP's time complexity did not permit us to run them in a reasonable amount of time

[1]Dataset created by de Campos et al. (2000) — see http://archive.ics.uci.edu/ml/datasets/Cardiotocography

with our computational resources.

Furthermore, we would like to point out that *MUCCA* is not only "asymptotically faster", but we can observe it also from the running times. For example, the average running times of our experiments on USPS (using a single core GNU/Linux VPS): *MUCCA* 6s, 11**MUCCA* 17s, GTG-ESS 310s, LABPROP 175s with train size 5% and 1530s with train size 0.5%.These values include reading data from HDD, the values are avgeraged over 10 runs and different training sizes. The value regarding *MUCCA* include the generation of spanning trees (RST). *MUCCA* and GTG-ESS have a small variance across training sets sizes.

6. Conclusions and future work

We introduced a novel scalable algorithm for multiclass node classification in arbitrary weighted graphs. Our algorithm is motivated within a game theoretic framework, where test labels are expressed as the Nash equilibrium of a certain game. In practice, MUCCA works well even on binary problems against competitors like Label Propagation and SHAZOO that have been specifically designed for the binary setting.

Several open questions remain. For example, committees of MUCCA predictors work well but we do not know whether there are better ways to aggregate their predictions. Also, given their common game-theoretic background, it would be interesting to explore possible connections between committees of MUCCA predictors and GTG-ESS.

Acknowledgements

We would like to thank Nicolò Cesa-Bianchi and Claudio Gentile for many useful discussions. We also would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

Aumann, R.J. Subjectivity and correlation in randomized strategies. In *Journal of Mathematical Economics*, 1974.

Blum, A. and Chawla, S. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th International Conference on Machine Learning.* Morgan Kaufmann, 2001.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, Learning and Games.* Cambidge University Press, 2006.

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

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

Cesa-Bianchi, N., Gentile, C., Vitale, F., and Zappella, G. See the tree through the lines: The shazoo algorithm. In *Advances in Neural Information Processing Systems 25*, 2011.

Costa, M., L.Letocart, and Roupin, F. Minimal multicut and maximal integer multiflow: a survey. In *European Journal of Operational Research.* Elsevier, 2005.

Dalhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., and Yannakakis, M. The complexity of multiway cuts. In *SIAM Journal of Computers*, 1994.

de Campos, D. Ayres, Bernardes, J., Garrido, A., de S, J. Marques, and Pereira-Leite, L. Sisporto 2.0 a program for automated analysis of cardiotocograms. In *Journal of Matern Fetal Med*, 2000.

Erdem, A. and Pelillo, M. Graph transduction as a noncooperative game. In *Graph-base Representation in Pattern Recognition.* Lecture Notes in Computer Science, 2011.

Herbster, M. and Pontil, M. Prediction on a graph with a perceptron. In *Advances in Neural Information Processing Systems 20.* MIT Press, 2006.

Herbster, M., Pontil, M., and Rojas-Galeano, S. Fast prediction on a tree. In *Advances in Neural Information Processing Systems 22.* MIT Press, 2009.

Kleinberg, J. and Tardos, E. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *Proceedings of the 40st Annual Symposium on the Foundations of Computer Science*, 1999.

Nash, J. Non-cooperative games. In *The Annals of Mathematics*, 1951.

Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V.V. *Algorithmic Game Theory.* Cambidge University Press, 2008.

Weibull, A. *Evolutionary Game Theory.* MIT Press, 1995.

Wilson, D.B. Generating random spanning trees more quickly than the cover time. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pp. 296–303. ACM Press, 1996.

Zhang, Y., Huang, K., and Liu, C. Fast and robust graph-based transductive learning via minimum tree cut. In *IEEE International Conference on Data Mining (ICDM)*, 2011.

Zhu, X., Ghahramani, Z., and Lafferty, J. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.