Peer-to-Peer Multi-Class Boosting*

István Hegedűs[1], Róbert Busa-Fekete[1,2], Róbert Ormándi[1], Márk Jelasity[1], and Balázs Kégl[2,3]

[1] Research Group on AI, Hungarian Acad. Sci. and Univ. of Szeged, Hungary
{ihegedus,busarobi,ormandi,jelasity}@inf.u-szeged.hu
[2] Linear Accelerator Laboratory (LAL), University of Paris-Sud, CNRS Orsay, 91898, France
[3] Computer Science Laboratory (LRI), University of Paris-Sud, CNRS and INRIA-Saclay, 91405 Orsay, France
balazs.kegl@gmail.com

Abstract. We focus on the problem of data mining over large-scale fully distributed databases, where each node stores only one data record. We assume that a data record is never allowed to leave the node it is stored at. Possible motivations for this assumption include privacy or a lack of a centralized infrastructure. To tackle this problem, earlier we proposed the generic gossip learning framework (GoLF), but so far we have studied only basic linear algorithms. In this paper we implement the well-known boosting technique in GoLF. Boosting techniques have attracted growing attention in machine learning due to their outstanding performance in many practical applications. Here, we present an implementation of a boosting algorithm that is based on FilterBoost. Our main algorithmic contribution is a derivation of a pure online multi-class version of FilterBoost, so that it can be employed in GoLF. We also propose improvements to GoLF, with the aim of maximizing the diversity of the evolving models gossiped in the network, a feature that we show to be important. We evaluate the robustness and the convergence speed of the algorithm empirically over three benchmark databases. We compare the algorithm with the sequential AdaBoost algorithm and we test its performance in a failure scenario involving message drop and delay, and node churn.

Keywords: P2P, gossip, multi-class classification, boosting, FilterBoost

1 Introduction

Making data analysis possible in fully distributed systems via data mining tools has been an important research direction in the past decade. Tasks such as in-

* The original publication is available at www.springerlink.com. In Proc. Euro-Par 2012, Springer LNCS 7484 pp. 389–400, doi:10.1007/978-3-642-32820-6_39. M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was partially supported by the FET programme FP7-COSI-ICT of the European Commission through project QLectives (grant no.: 231200) and by the ANR-2010-COSI-002 grant of the French National Research Agency.

formation retrieval, recommendations, detecting spam, vandalism and intrusion require sophisticated models that are based on large amounts of data. This data is often generated in a fully distributed fashion on routers, PCs, smart phones or sensor nodes. In many cases local data cannot be collected centrally due to privacy constrains or due to the lack of computing infrastructure.

In this paper we are concerned with the scenario in which there is a very large number of nodes, all of which store small amounts of data, such as personal profiles or recent sensor readings. In our previous work, we have proposed the gossip learning framework (GoLF) for data mining in such environments [19,20]. The basic idea is that models perform random walks in the network, while being improved by an arbitrary *online learning* method. Convergence can be improved significantly if nodes combine the models that pass through them, or if they use other techniques such as voting. In this framework we have so far only studied learning linear models.

In this paper we develop a boosting algorithm, which proves the viability of gossip learning also for implementing state-of-the-art machine learning algorithms. In a nutshell, a boosting algorithm constructs a classifier in an incremental fashion by adding simple classifiers (that is, weak classifiers) to a pool. The weighted vote of the classifiers in the pool determines the final classification.

Our contributions are the following. First, to enable P2P boosting via gossip, we derive a purely online multi-class boosting algorithm that can be proven to minimize a certain negative log likelihood function. We also introduce efficient multi-class weak learners to be used by the online boosting algorithm. Second, we improve GoLF to make sure that the diversity of the models in the network is preserved. This makes it meaningful to spread the current best model in the network; a technique we propose to improve local prediction performance. Finally, we perform simulation experiments where we study our algorithm under extreme message drop, message delay and node churn to prove its robustness.

2 System Model and Data Distribution

Our system model is a network of computers (peers). Each node in the network has a unique network address and can communicate with other nodes through messages if the address of the target node is locally available. We also assume that a *peer sampling service* is available that provides addresses of random available peers at any time. Here we use Newscast [26] as our peer sampling service. Messages can be delayed or dropped, moreover, new nodes can join and leave the network without any warning. We assume that when a node rejoins the network it has the same state as at the time of going offline.

Regarding data distribution, we assume that the data records are distributed horizontally, that is, all the nodes store full records. At the same time, all the nodes store only very few records, perhaps only a single record. This excludes the possibility of any local statistical processing of the data. Another important assumption is that the data never leave the nodes, that is, it is not allowed to collect the data centrally due to privacy or infrastructural constraints.

3 Background and Related Work

The problem we tackle in this paper is *supervised classification* that can be formally defined as follows. We are given a training database in the form of a set of training instances. Each training instance consists of a feature vector and a corresponding class label. Let us denote this training dataset by $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\} \subset \mathbb{R}^d \times \{-1, +1\}^K$, where d is the *dimension* of the problem and K defines the number of classes. The goal of the classification problem is to find a function $\mathbf{f} : \mathbb{R}^d \to \{-1, +1\}^K$ that can correctly classify *any samples*, including those not in the training set, with high probability *(generalization)*. In *multi-class* classification problems—where $K > 2$—one and only one of the elements of $\mathbf{y}_i$ is $+1$, whereas in *multi-label* (or *multi-task*) classification $\mathbf{y}_i$ is arbitrary, meaning that the observation $\mathbf{x}_i$ can belong to several classes at the same time. In the former case we will denote the index of the correct class by $\ell(\mathbf{x}_i)$. In classical multi-class classification the elements of $\mathbf{f}(\mathbf{x})$ are treated as posterior scores corresponding to the labels, so the predicted label is $\widehat{\ell}(\mathbf{x}) = \arg\max_{\ell=1,\ldots,K} f_\ell(\mathbf{x})$ where $f_\ell(\mathbf{x})$ is the ℓth element of $\mathbf{f}(\mathbf{x})$. The function $\mathbf{f}$ is called the *model* of the data.

As mentioned before, in this paper we focus on online boosting in GoLF. A few proposals for online boosting algorithms are known. An online version of AdaBoost [11] is introduced in [8] that requires a random subset from the training data for each boosting iteration, and the base learner is trained on this small sample of the data. The algorithm has to sample the data according to a non-uniform distribution making it inappropriate for pure online training. A gradient–based online algorithm is presented in [3], which is an extension of Friedman's gradient–based framework [12]. However, their approach is for binary classification, and it is not obvious how it can be extended to multi-class problems. Another notable online approach is Oza's online algorithm [21] whose starting point is AdaBoost.M1 [10]. However, AdaBoost.M1 requires the base learning algorithm to achieve 50% accuracy for any distribution over the training instances. This makes it impractical in multi-class classification since most of the weak learners used as a base learner do not satisfy this condition.

We also discuss work related to fully distributed P2P data mining in general. We note that we do not overview the extensive literature of parallel machine learning algorithms because they have a completely different underlying system model and motivation. We do not discuss those distributed machine learning approaches either that assume the availability of sufficient local data to build models locally (a survey can be found in [22]).

One notable and relevant research direction is gossip–based algorithms where convergence to global functions over fully distributed data is achieved through local communication. Perhaps the simplest example is gossip–based averaging [14,16], where the gossip approach is extremely robust, scalable, and efficient. However, gossip algorithms support more sophisticated algorithms that compute more complex global functions. Examples include the EM algorithm [17], LDA [2] or PageRank [13]. Numerous other P2P machine learning algorithms have also been proposed, as in [18,25]. A survey of many additional ideas can be found in [7].

Algorithm 1 Skeleton of original GoLF learning protocol

```
1: currentModel ← initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     sendModel(p, currentModel)
6: procedure onReceiveModel(m)
7:     m.updateModel(x, y)
8:     currentModel ← m
```

This work builds on the Gossip Learning Framework (GoLF) [19,20], which offers an abstraction to implement a wide range of machine learning algorithms.

The skeleton of GoLF is shown in Alg. 1. This algorithm runs on each node. The algorithm consists of an *active loop* that runs periodically and an event handler (onReceiveModel) which is called when a new model arrives. The models take random walks over the network by selecting a random node (line 4) and jumping there (line 5). Procedure onReceivedModel updates the received model using the training sample stored by the node (line 7, where x and y represent a training example and the corresponding class label, respectively). It then stores the model as the current model (line 8). In this skeleton the model is an abstract class which provides the update possibility. We note that models can also be combined [20] or they can interact through ensemble learning techniques (like voting) [19], which results in a substantial performance improvement. Regarding model interaction, additional details will be given later in relation to the boosting algorithm.

4 Multi-Class Online FilterBoost

This section introduces our main contribution, a multi-class online boosting algorithm that can be applied in GoLF. We build on FilterBoost [5] where the main idea is to filter (sample) the training examples in each boosting iteration and to give the base learner only this smaller, filtered subset of the original training dataset, leading to fast base learning. The performance of the base classifier is also estimated on an additional random subset of the training set resulting in further improvement in speed.

Our formulation of the FilterBoost algorithm is given as Alg. 2. This is not yet in a form to be applied in GoLF, but the transformation is trivial as discussed in Section 6. This fully online formulation is equivalent to FilterBoost, except that it handles multi-class problems as well. To achieve this, while ensuring that the algorithm can still be theoretically proven to converge, our key contribution is the derivation of a new weight formula calculated in line 19. First we introduce this formula, then we explain Alg. 2 in more detail.

A boosting algorithm can be thought of as a minimization algorithm of an appropriately defined target function over the space of models. The target function is related to the classification error over the training dataset. The key idea is that we select an appropriate target function that will allow us to both derive an appropriate weight, as well as argue for convergence. Inspired by the logistic regression approach of [6], we will use the following negative log likelihood function as our target function:

Algorithm 2 FilterBoost(Init(), Update($\cdot,\cdot,\cdot,\cdot$), T, C)

1: $\mathbf{f}^{(0)}(\mathbf{x}) \leftarrow 0$
2: **for** $t \leftarrow 1 \rightarrow T$ **do**
3: $\quad C_t \leftarrow C \log(t+1)$
4: $\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \textsc{Init}()$
5: $\quad$ **for** $t' \leftarrow 1 \rightarrow C_t$ **do** ▷ Online base learning
6: $\quad\quad (\mathbf{x},\mathbf{y},\mathbf{w}) \leftarrow \textsc{Filter}(\mathbf{f}^{(t-1)}(\cdot))$ ▷ Draw a weighted random instance
7: $\quad\quad \mathbf{h}^{(t)}(\cdot) \leftarrow \textsc{update}(\mathbf{x},\mathbf{y},\mathbf{w},\mathbf{h}^{(t)}(\cdot))$
8: $\quad \gamma \leftarrow 0, W \leftarrow 0$
9: $\quad$ **for** $t' \leftarrow 1 \rightarrow C_t$ **do** ▷ Estimate the edge on a filtered data
10: $\quad\quad (\mathbf{x},\mathbf{y},\mathbf{w}) \leftarrow \textsc{Filter}(\mathbf{f}^{(t-1)}(\cdot))$ ▷ Draw a weighted random instance
11: $\quad\quad \gamma \leftarrow \gamma + \sum_\ell^K w_\ell h_\ell^{(t)}(\mathbf{x}) y_\ell$, $W \leftarrow W + \sum_\ell^K w_\ell$
12: $\quad \gamma \leftarrow \gamma / W$ ▷ Normalize the edge
13: $\quad \alpha^{(t)} \leftarrow \frac{1}{2}\log\frac{1+\gamma}{1-\gamma}$
14: $\quad \mathbf{f}^{(t)}(\cdot) = \mathbf{f}^{(t-1)}(\cdot) + \alpha^{(t)}\mathbf{h}^{(t)}(\cdot)$
15: **return** $\mathbf{f}^{(T)}(\cdot) = \sum_{t=1}^T \alpha^{(t)}\mathbf{h}^{(t)}(\cdot)$
16: **procedure** Filter($\mathbf{f}(\cdot)$)
17: $\quad (\mathbf{x},\mathbf{y}) \leftarrow \textsc{RandomInstance}()$ ▷ Draw random instance
18: $\quad$ **for** $\ell \leftarrow 1 \rightarrow K$ **do**
19: $\quad\quad w_\ell \leftarrow \frac{\exp\left(f_\ell(\mathbf{x}) - f_{\ell(\mathbf{x})}(\mathbf{x})\right)}{\sum_{\ell'=1}^K \exp\left(f_{\ell'}(\mathbf{x}) - f_{\ell(\mathbf{x})}(\mathbf{x})\right)}$
20: $\quad$ **return** $(\mathbf{x},\mathbf{y},\mathbf{w})$

$$R_{\mathrm{L}}(\mathbf{f}) = -\sum_{i=1}^n \ln \frac{\exp\left(f_{\ell(\mathbf{x}_i)}(\mathbf{x}_i)\right)}{\sum_{\ell'=1}^K \exp\left(f_{\ell'}(\mathbf{x}_i)\right)} = \sum_{i=1}^n \ln\left[1 + \sum_{\ell\neq\ell(\mathbf{x}_i)}^K \exp\left(f_\ell(\mathbf{x}_i) - f_{\ell(\mathbf{x}_i)}(\mathbf{x}_i)\right)\right] \tag{1}$$

Note that the FilterBoost algorithm returns a vector-valued classifier $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^K$. The rest of the definitions and notations were introduced in Section 3.

FilterBoost builds the final classifier $\mathbf{f}$ as a weighted sum of *base classifiers* $\mathbf{h}^{(t)} : \mathbb{R}^d \rightarrow \{-1,+1\}^K$ returned by a *base learner* algorithm which has to be able to handle weighted training data. The class-related weight vector assigned to $\mathbf{x}_i$ in iteration t is denoted by $\mathbf{w}_i^{(t)}$ and its ℓth element is denoted by $w_{i,\ell}^{(t)}$. It can be shown that selecting $w_{i,\ell}^{(t)}$ so that it is proportional to the output of the current strong classifier

$$w_{i,\ell}^{(t)} = \frac{\exp\left(f_\ell^{(t)}(\mathbf{x}_i) - f_{\ell(\mathbf{x}_i)}^{(t)}(\mathbf{x}_i)\right)}{\sum_{\ell'=1}^K \exp\left(f_{\ell'}^{(t)}(\mathbf{x}_i) - f_{\ell(\mathbf{x}_i)}^{(t)}(\mathbf{x}_i)\right)}. \tag{2}$$

ensures that our target function in (1) will decrease in each boosting iteration. The proof is outlined in the Appendix.

The pseudocode of FilterBoost is shown in Alg. 2. Here, the algorithm is implemented according to the practical suggestions given in [5]: first, the number of randomly selected instances is $C\log(t+1)$ in the tth iteration (where C is a constant parameter), and second, in the Filter method the instances are first randomly selected then re-weighted based on their scores given by $\mathbf{f}^{(t)}(\cdot)$. Procedure Init() initializes the parameters of the base classifier (line 4), and Update($\cdot,\cdot,\cdot,\cdot$) updates (line 7) the parameter of the base classifier using the

current training instance $\mathbf{x}$ given by FILTER$(\cdot)$. The input parameter T is the number of iterations, and C controls the number of instances used in one boosting iteration. $\alpha^{(t)}$ is the base coefficient, $\mathbf{h}^{(t)}(\cdot)$ is the vector-valued base classifier, and $\mathbf{f}^{(T)}(\cdot)$ is the final (strong) classifier. Procedure RANDOMINSTANCE() selects a random instance from the training data $\mathbf{X}, \mathbf{Y}$.

Let us point out that there is no need to store more than one training instance anywhere during execution. Second, the algorithm does not need any global information about the training data, such as the size, so this implementation can be readily applied in a pure online environment.

5 Multi-Class Online Base Learning

For the online version of FILTERBOOST, we need to propose online base learners as well. In FILTERBOOST, for theoretical reasons, the base classifiers are restricted to output discrete predictions in $\{-1,+1\}^K$ and, in addition, they have to minimize the weighted exponential loss

$$E\big(\mathbf{h}, \mathbf{f}^{(t)}\big) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell}^{(t)} \exp\big(-h_\ell(\mathbf{x}_i) y_{i,\ell}\big). \tag{3}$$

We follow this approach and, in addition, we build on our base learning framework [15] and assume that the base classifier $\mathbf{h}(\mathbf{x})$ is vector-valued and represented as $\mathbf{h}_\Theta(\mathbf{x}) = \mathrm{sign}(\mathbf{v}\varphi_\Theta(\mathbf{x}))$, parameterized by $\mathbf{v} \in \mathbb{R}^K$ (the *vote vector*), and $\varphi_\Theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$, a *scalar* base classifier parameterized by Θ. The coordinate-wise sign function is defined as $\mathrm{sign} : \mathbb{R}^K \to \{-1,+1\}^K$. In this framework, learning consists of tuning Θ and $\mathbf{v}$ to minimize the weighted exponential loss (3).

Since it is hard to optimize the non-differentiable function $\mathbf{h}_\Theta$ even in batch mode, we take into account only $\widehat{\mathbf{h}}_\Theta(\mathbf{x}) = \mathbf{v}\varphi_\Theta(\mathbf{x})$. This approach is heuristic as it is hard to say anything about the relation between $E\big(\mathbf{h}_\Theta, \mathbf{f}^{(t)}\big)$ and $E\big(\widehat{\mathbf{h}}_\Theta, \mathbf{f}^{(t)}\big)$, but in practice this base learning approach performs quite well.

Since $\varphi_\Theta(\cdot)$ is differentiable, the stochastic gradient descent (SGD) [4] algorithm provides a convenient way to train the base learner in an online fashion. The SGD algorithm updates the parameters iteratively based on one training instance at a time. Let us denote $Q(\mathbf{x}, \mathbf{y}, \mathbf{w}, \mathbf{v}, \Theta) = \sum_{\ell=1}^{K} w_\ell \exp\big(-y_\ell v_\ell \varphi_\Theta(\mathbf{x})\big)$. Then the gradient based parameter update can be calculated as follows:

$$\Theta^{(t'+1)} \leftarrow \Theta^{(t')} + \gamma^{(t')} \nabla_\Theta Q(\mathbf{x}, \mathbf{y}, \mathbf{w}, \mathbf{v}, \Theta) \tag{4}$$

$$\mathbf{v}^{(t'+1)} \leftarrow \mathbf{v}^{(t')} + \gamma^{(t')} \nabla_{\mathbf{v}} Q(\mathbf{x}, \mathbf{y}, \mathbf{w}, \mathbf{v}, \Theta) \tag{5}$$

This update rule can be used in line 7 of FILTERBOOST to update the base classifier. A simple decision stump or ADALINE [27] can be easily accommodated to this multi-class base learning framework. In the following we derive the update rules for a decision stump, that is, a one-decision two-leaf decision tree having the form

$$\varphi_{j,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } x^{(j)} \geq b, \\ -1 & \text{otherwise,} \end{cases} \tag{6}$$

where j is the index of the selected feature and b is the decision threshold. Since $\varphi_{j,b}(\mathbf{x})$ is not differentiable with respect to b, we decided to approximate it by the differentiable sigmoidal function, whose parameters can be tuned using SGD. The sigmoidal function can be written as

$$s_{j,\theta}(\mathbf{x}) = s_{j,(c,d)}(\mathbf{x}) = \frac{1}{1 + \exp\big(-cx^{(j)} - d\big)}.$$

where $\Theta = (c, d)$. And $\varphi_{j,b}(\cdot)$ can be approximated by $\varphi_{j,b}(\mathbf{x}) \approx 2s_{j,\theta}(\mathbf{x}) - 1$. Then the weighted exponential loss of this so-called *sigmoidal decision stump* for a single instance can be written as

$$Q_j = Q_j\left(\mathbf{x}, \mathbf{y}, \mathbf{w}, \mathbf{v}, \Theta\right) = \sum_{\ell=1}^{K} w_\ell \exp\left(-v_\ell \left(2s_{j,\theta}(\mathbf{x}) - 1\right) y_\ell\right)$$

and its partial derivatives are

$$\begin{aligned}
\frac{\partial Q_j}{\partial v_\ell} &= -\exp\left(-v_\ell \left(2s_{j,\theta}(\mathbf{x}) - 1\right) y_\ell\right) w_\ell \left(2s_{j,\theta}(\mathbf{x}) - 1\right) y_\ell \\
\frac{\partial Q_j}{\partial c} &= -2\sum_{\ell=1}^{K} \exp\left(-v_\ell \left(2s_{j,\theta}(\mathbf{x}) - 1\right) y_\ell\right) w_\ell v_\ell y_\ell x^{(j)} s_{j,\theta}(\mathbf{x}) \left(1 - s_{j,\theta}(\mathbf{x})\right) \\
\frac{\partial Q_j}{\partial d} &= -2\sum_{\ell=1}^{K} \exp\left(-v_\ell \left(2s_{j,\theta}(\mathbf{x}) - 1\right) y_\ell\right) w_\ell v_\ell y_\ell s_{j,\theta}(\mathbf{x}) \left(1 - s_{j,\theta}(\mathbf{x})\right)
\end{aligned}$$

The initial value of c and d were set to 1 and 0, respectively (line 4 of Alg. 2).

So far, we implicitly assumed that the index of feature j is given. To choose j, we trained sigmoidal decision stumps in parallel for each feature and we estimated the edge of each of them using the sequential training data as $\widehat{\gamma}_j = \sum_{t'=1}^{C_t} \sum_{\ell=1}^{K} w_{t',\ell} y_{t',\ell} \mathrm{sign}\big(v_\ell^{(t')} \varphi_{j,\Theta_j^{(t')}}(\mathbf{x}_{t'})\big)$. Finally, we chose the feature with the highest edge estimate $j^* = \arg\max_j \widehat{\gamma}_j$.

In every boosting iteration we also train a *constant learner* (also known as y-intercept) and use it if its edge is higher than the edge of the best decision stump we found. The output of the constant learner does not depend on the input vector $\mathbf{x}$, that is $\varphi(\cdot) \equiv 1$, in other words it returns the vote vector $\mathbf{v}$ itself. Thus only $\mathbf{v}$ has to be learnt but this can be done easily by calculating the classwise edge $v_\ell = \sum_{t'=1}^{C_t} w_{t',\ell} y_{t',\ell}$.

6 GoLF Boosting

In order to adapt Alg. 2 to GoLF (Alg. 1), we need to define the permanent state of the FilterBoost model class, and we need to provide an implementation of the updateModel method. This is rather straightforward: the model instance has to store the the actual strong learner $\mathbf{f}^{(t)}$ as well as the state of the inner part of the two for loops in Alg. 2 so that updateModel could simulate these loops every time a new sample is processed.

Algorithm 3 Diversity Preserving GoLF

```
1:  currentModel ← initModel()
2:  modelQueue.add(currentModel)
3:  counter ← 0
4:  loop
5:      wait(Δ)
6:      if modelQueue.isEmpty() then
7:          if counter = 10 then
8:              p ← selectPeer()
9:              sendModel(p, currentModel)
10:             counter ← 0
11:         else
12:             counter ← counter + 1
13:     else
14:         for all m ∈ modelQueue do
15:             p ← selectPeer()
16:             sendModel(p, m)
17:             modelQueue.remove(m)
18:         counter ← 0

19: procedure onReceiveModel(m)
20:     m.updateModel(x, y)
21:     modelQueue.add(m)
22:     currentModel ← m
```

This way, every model that is performing a random walk is theoretically guaranteed to converge so long as we assume that peer sampling works perfectly. However, there is a catch. Since in each iteration some nodes will receive more than one model, while others will not receive any, and since the number of models in the network is kept constant if there is no failure (since in each iteration all the nodes send exactly one model) it is clear that the *diversity* of models will decrease. That is, some models get replicated, while others "die out". Introducing failure makes things a lot worse, since we can lose models due to message loss, delay, and churn as well, which speeds up homogenization. This is a problem, because diversity is important when we want to apply techniques such as combination or voting [19,20]. Without diversity these important techniques are guaranteed not to be effective.

The effects of decreasing diversity are negligible during the timespan of a few gossip cycles, but a boosting algorithm needs a relatively large number of cycles to converge (which is not a problem, since the point of boosting is not speed, but classification quality). So we need to tackle the loss of diversity. We propose Alg. 3 to deal with this problem.

This protocol works as follows. A node sends models in an active cycle (line 4) only in two cases: it sends the last received model if there was no incoming model until 10 active cycles (line 6), otherwise it sends all of the models received since the last cycle (line 13). If there is no failure, then this protocol is guaranteed to keep the diversity of models, since all the models in the network will perform independent random walks. Due to the Poisson distribution of the number of incoming models in one cycle, the probability of bottlenecks is diminishing, and for the same reason the probability that a node does not receive messages for 10 cycles is also practically negligible.

If the network experiences message drop failures or churn, then the number of models circulating in the network will converge to a smaller value due to the 10 cycle waiting time, and the diversity can also decrease, since after 10 cycles a model gets replicated in line 9. Interestingly, this is actually useful because if the diversity is low, it makes sense to circulate fewer models and to wait most of the time, since information is redundant anyway. Besides, with reliable communication channels that eliminate message drop (but still allow for delay), diversity can still be maintained.

Table 1. The main properties of the data sets, and the prediction errors of the baseline algorithms.

	CTG	PenDigits	Segmentation
Training set size	1,701	7494	2100
Test set size	425	3,492	210
Number of features	21	16	19
Class labels	1325/233/143	10 classes (uniform)	7 classes (uniform)
AdaBoost (DS)	0.109347	0.060715	0.069048
FilterBoost (DS, C30)	0.094062	0.071657	0.062381

Finally, note that if there is no failure, Alg. 3 has the same total message complexity as Alg. 1 except for the extremely rare messages sent in line 4. In case of failure, the message complexity decreases as a function of failure rate; however, the remaining random walks do not get slower relative to Alg. 1, so the convergence rate remains the same on average, at least if no model-combination techniques are used.

7 Experimental Results

In our experiments we examined the performance of our proposed algorithm as a function of gossip cycles, which is about the same as the number of training samples seen by any particular model. To validate the algorithm, we compared it with three baseline multi-class boosting algorithms, all using the same decision stump (DS) weak learner. The first one is the multi-class version of the well known AdaBoost [24] algorithm, the second one is the original FilterBoost [5] method implemented for a single processor, with the setting $C = 30$, and the third one is the online version of FilterBoost (Alg. 2). We used three multi-class classification benchmark datasets to evaluate our method, namely the CTG, the PenDigits and the Segmentaion databases. These were taken from the UCI repository [9] and have different size, number of features, class distributions and characteristics. The basic properties of the datasets can be found in Table 1.

In the P2P experiments we used the PeerSim [23] simulation environment to model message *delay, drop* and peer *churn*. We used two scenarios: a perfect network without any delay, drop or churn; and a scenario with heavy failure where the message delay was drawn uniformly at random from the interval $[\Delta; 10\Delta]$, a message was dropped with a probability of 0.5 and the online/offline session lengths of peers were modeled using a real P2P bittorrent trace [1]. As our performance metric, we applied the well known *0-1 error* (or error rate), which is the proportion of test instances that were incorrectly classified.

Figure 1 illustrates the effect of parameter C. Larger values result in slower convergence but better eventual performance. The setting $C = 30$ represents a good tradeoff in these datasets, so from now on we fix this value.

We compared our online boosting algorithm to baseline algorithms as can be seen in Figure 2 (left hand side). The figure shows that the algorithms converge

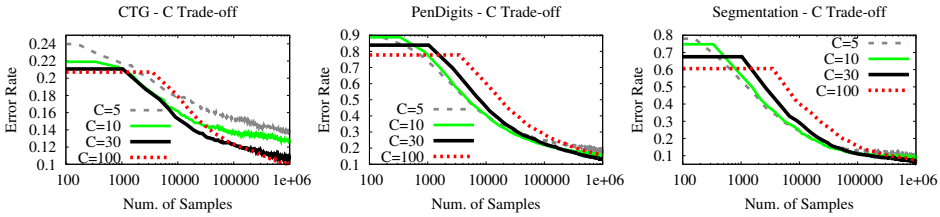

Fig. 1. The effect of parameter C in online FilterBoost (Alg. 2).

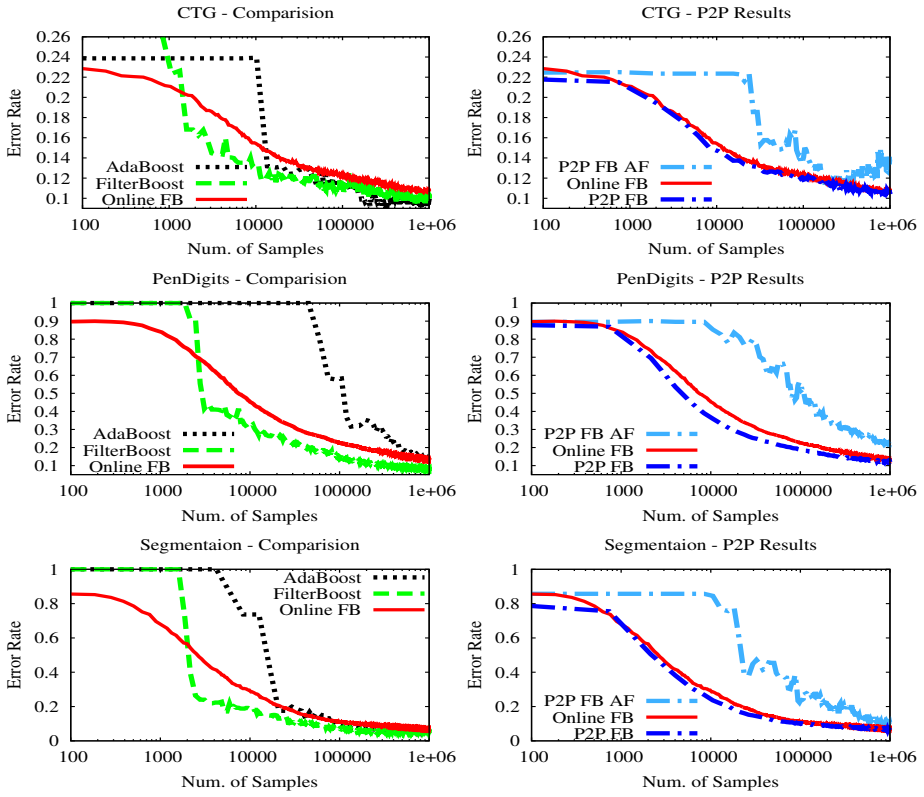

Fig. 2. Comparison of boosting algorithms (left column) and P2P simulations (right column). FB and AF stand for FilterBoost and the "all failures" scenario, respectively.

to a similar error rate, which was expected. Moreover, our online FilterBoost converges faster than the AdaBoost algorithm and it has almost the same convergence rate as that for the sequential FilterBoost method. Note that since two of these algorithms are not online, we had to approximate the number of (not necessarily different) training samples used in one boosting iteration. We used a lower bound to be conservative.

In our P2P evaluations of GoLF Boosting we used the mean error rate of 100 randomly selected nodes in the network to approximate the performance of the algorithm. Figure 2 (right hand side) shows that without failure the perfor-

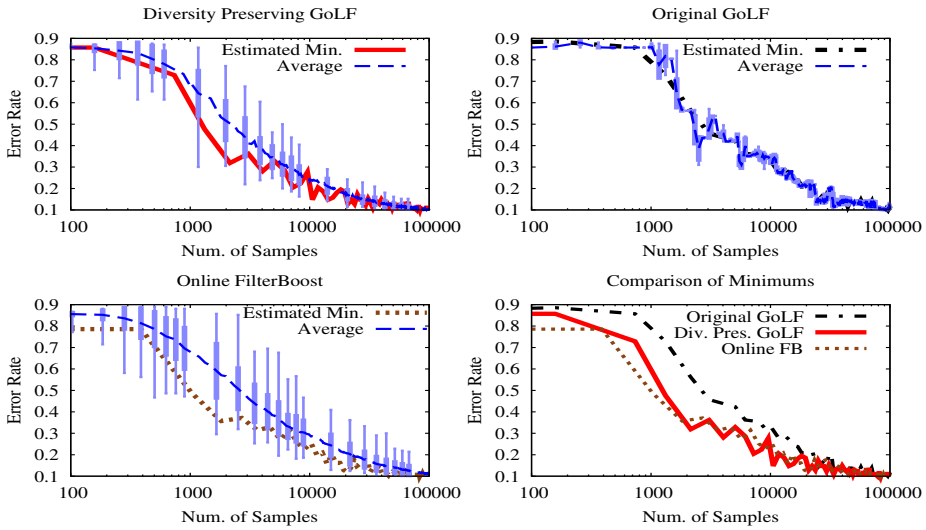

Fig. 3. The improvement due to estimating the best model based on training performance. The Segmentation dataset is shown.

mance is very similar to that of our online FilterBoost algorithm. Moreover, in the extreme failure scenario, the algorithm still converges to the same error rate, although with a delay. This delay can be accounted for using a heuristic argument: since message delay in itself represents a slowdown of a factor of 5 on average, message drop and churn contributes approximately another factor of 2.

Finally, we demonstrate a novel way of exploiting model diversity (see Section 6): through gossip-based minimization one can spread the model with the *best training performance*, thus the best model can be made available to all nodes at all times. Figure 3 demonstrates this technique for different algorithms. We include results over the segmentation database only, the other two datasets produce similar results.

The top left plot shows results with GoLF Boosting. It can be seen that the best model based on training performance is not necessarily the best over the test set, but it is reasonably good, and results in a speedup of about a factor of 2. The top right plot belongs to the original GoLF implementation (Alg. 1). Due to the complete lack of diversity, the best model's performance is almost identical to the average one. The bottom left plot is a baseline experiment that represents the case with the maximal possible diversity, based on 100 completely independent runs of the online FilterBoost algorithm. Finally, the bottom right plot collects the most interesting curves from the other three plots allowing a better comparison.

8 Conclusions

We demonstrated that the GoLF is suitable for the implementation of multi-class boosting. The significance of this result is that boosting is a state-of-the-art machine learning technique from the point of view of the quality of the learned models, which is now available in the P2P system model with fully distributed data. To achieve this, we proposed a modification of FilterBoost that allows it to learn multi-class models in a purely online fashion, and we proved theoretically that the resulting algorithm optimizes a suitably defined negative log likelihood measure. Our experimental results demonstrate the robustness of the method. We also identified the lack of model diversity as a potential problem with GoLF. We provided a solution that was demonstrated to be effective in preserving the difference between the best model and the average models; this allowed us to propose spreading the best model as a way to benefit from the large number of models in the network.

References

1. Filelist. `http://www.filelist.org` (2005)
2. Asuncion, A.U., Smyth, P., Welling, M.: Asynchronous distributed estimation of topic models for document analysis. Statistical Methodology 8(1), 3 – 17 (2011)
3. Babenko, B., Yang, M., Belongie, S.: A family of online boosting algorithms. In: Computer Vision Workshops (ICCV Workshops). pp. 1346–1353 (2009)
4. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Intl. Conf. on Computational Statistics. vol. 19, pp. 177–187 (2010)
5. Bradley, J., Schapire, R.: FilterBoost: Regression and classification on large datasets. In: Advances in Neural Information Processing Systems. vol. 20. The MIT Press (2008)
6. Collins, M., Schapire, R., Singer, Y.: Logistic regression, AdaBoost and Bregman distances. Machine Learning 48, 253–285 (2002)
7. Datta, S., Bhaduri, K., Giannella, C., Wolff, R., Kargupta, H.: Distributed data mining in peer-to-peer networks. IEEE Internet Comp. 10(4), 18–26 (July 2006)
8. Fan, W., Stolfo, S.J., Zhang, J.: The application of AdaBoost for distributed, scalable and on-line learning. In: Proc. 5th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining. pp. 362–366 (1999)
9. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
10. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Machine Learning: Proc. Thirteenth Intl. Conf. pp. 148–156 (1996)
11. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. of Comp. and Syst. Sci. 55, 119–139 (1997)
12. Friedman, J.: Stochastic gradient boosting. Computational Statistics and Data Analysis 38(4), 367–378 (2002)
13. Jelasity, M., Canright, G., Engø-Monsen, K.: Asynchronous distributed power iteration with gossip-based normalization. In: Euro-Par 2007. LNCS, vol. 4641, pp. 514–525. Springer (2007)
14. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Trans. on Computer Systems 23(3), 219–252 (August 2005)

15. Kégl, B., Busa-Fekete, R.: Boosting products of base classifiers. In: Intl. Conf. on Machine Learning. vol. 26, pp. 497–504. Montreal, Canada (2009)
16. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03). pp. 482–491. IEEE Computer Society (2003)
17. Kowalczyk, W., Vlassis, N.: Newscast EM. In: 17th Advances in Neural Information Processing Systems (NIPS). pp. 713–720. MIT Press, Cambridge, MA (2005)
18. Luo, P., Xiong, H., Lü, K., Shi, Z.: Distributed classification in peer-to-peer networks. In: Proc. 13th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining (KDD'07). pp. 968–976. ACM, New York, NY, USA (2007)
19. Ormándi, R., Hegedűs, I., Jelasity, M.: Asynchronous peer-to-peer data mining with stochastic gradient descent. In: Euro-Par 2011. LNCS, vol. 6852, pp. 528–540. Springer (2011)
20. Ormándi, R., Hegedüs, I., Jelasity, M.: Efficient p2p ensemble learning with linear models on fully distributed data. CoRR abs/1109.1396 (2011)
21. Oza, N., Russell, S.: Online bagging and boosting. In: Proc. Eighth Intl. Workshop on Artificial Intelligence and Statistics (2001)
22. Park, B.H., Kargupta, H.: Distributed data mining: Algorithms, systems, and applications. In: Ye, N. (ed.) The Handbook of Data Mining. CRC Press (2003)
23. PeerSim: http://peersim.sourceforge.net/
24. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine Learning 37(3), 297–336 (1999)
25. Siersdorfer, S., Sizov, S.: Automatic document organization in a P2P environment. In: Advances in Information Retrieval, LNCS, vol. 3936, pp. 265–276. Springer (2006)
26. Tölgyesi, N., Jelasity, M.: Adaptive peer sampling with newscast. In: Euro-Par 2009. LNCS, vol. 5704, pp. 523–534. Springer (2009)
27. Widrow, B., Hoff, M.E.: Adaptive Switching Circuits. In: 1960 IRE WESCON Convention Record. vol. 4, pp. 96–104 (1960)

Appendix

The second order expansion of multi-class negative log likelihood for fixed α and $\mathbf{h}(\mathbf{x}) = 0$ can be written as

$$R_{\mathrm{L}}\big(\mathbf{f}^{(t)} + \alpha\mathbf{h}\big) = \ln\left(1 + \sum_{\ell\neq\ell(\mathbf{x})}^{K} \mathcal{F}_{\ell}^{\mathbf{f}^{(t)}}(\mathbf{x})\right) - \sum_{\ell}^{K} \frac{\mathcal{F}_{\ell}^{\mathbf{f}^{(t)}}(\mathbf{x})}{\sum_{\ell'=1}^{K}\mathcal{F}_{\ell'}^{\mathbf{f}^{(t)}}(\mathbf{x})}\alpha y_\ell h_\ell(\mathbf{x}) + \frac{1}{2}\sum_{\ell=1}^{K} \frac{\alpha y_\ell\left(1+\sum_{\ell'\neq\ell}\mathcal{F}_{\ell'}^{\mathbf{f}^{(t)}}(\mathbf{x})\right) - \alpha^2 \overbrace{y_\ell^2 h_\ell^2(\mathbf{x})}^{=1}\mathcal{F}_{\ell}^{\mathbf{f}^{(t)}}(\mathbf{x})}{1+\sum_{\ell'\neq\ell}\mathcal{F}_{\ell'}^{\mathbf{f}^{(t)}}(\mathbf{x})}$$

where $\mathcal{F}_{\ell}^{\mathbf{f}^{(t)}}(\mathbf{x}) = \exp\big(f_\ell^{(t)}(\mathbf{x}) - f_{\ell(\mathbf{x})}^{(t)}(\mathbf{x})\big)$. Let us note that the last term does not depend on $\mathbf{h}(\cdot)$, consequently minimizing this approximation of $R_{\mathrm{L}}\big(\mathbf{f}^{(t)} + \alpha\mathbf{h}\big)$ with respect to $\mathbf{h}(\mathbf{x})$ is equivalent to maximizing the weighted accuracy and the weight of the ℓth label is

$$w_\ell^{(t)} = \frac{\mathcal{F}_{\ell}^{\mathbf{f}^{(t)}}(\mathbf{x})}{\sum_{\ell'=1}^{K}\mathcal{F}_{\ell'}^{\mathbf{f}^{(t)}}(\mathbf{x})} = \frac{\exp\big(f_\ell^{(t)}(\mathbf{x}) - f_{\ell(\mathbf{x})}^{(t)}(\mathbf{x})\big)}{\sum_{\ell'=1}^{K}\exp\big(f_{\ell'}^{(t)}(\mathbf{x}) - f_{\ell(\mathbf{x})}^{(t)}(\mathbf{x})\big)}$$