Gossip-based Learning under Drifting Concepts in Fully Distributed Networks

István Hegedűs, Róbert Ormándi, Márk Jelasity

University of Szeged, Hungary

Motivation

- Explosive growth of smart phone platforms, and
- Availability of sensor and other contextual data
- Makes collaborative data mining possible
  - Health care: following and predicting epidemics, personal diagnostics
  - Smart city: traffic optimization, accident forecasting
  - (predicting earthquakes, financial applications, etc)
- P2P networks, grid, etc, are also relevant platforms

P2P system model

- Large number (millions or more) computers (nodes)
- Packet switched communication
  - Every node has an address
  - Any node can send a message to any given address
    - **Not actually true: NATs, firewalls**
- Messages can be delayed or lost, nodes can crash (unreliable asynchronous communication)

Fully distributed data

- Horizontal data distribution
- Every node has very few records, we assume they have only one
- Drifting: data distribution can change
- We do not allow for moving data, only local processing (privacy preservation)
- We require that the models are cheaply available for all the nodes

In the Network or on Servers (cloud)?

- The cloud is flexible and scalable but it is not free and not public: business model needed
  - It is cheap but with LOTS of data and communication it will get expensive
- Privacy is a concern
- P2P is more limited in what it can do
  - But not as much as it seems at first!
- Smart phones (unlike motes) are increasingly powerful devices
- P2P and cloud hybrids possible (the network can act as a sensor!)

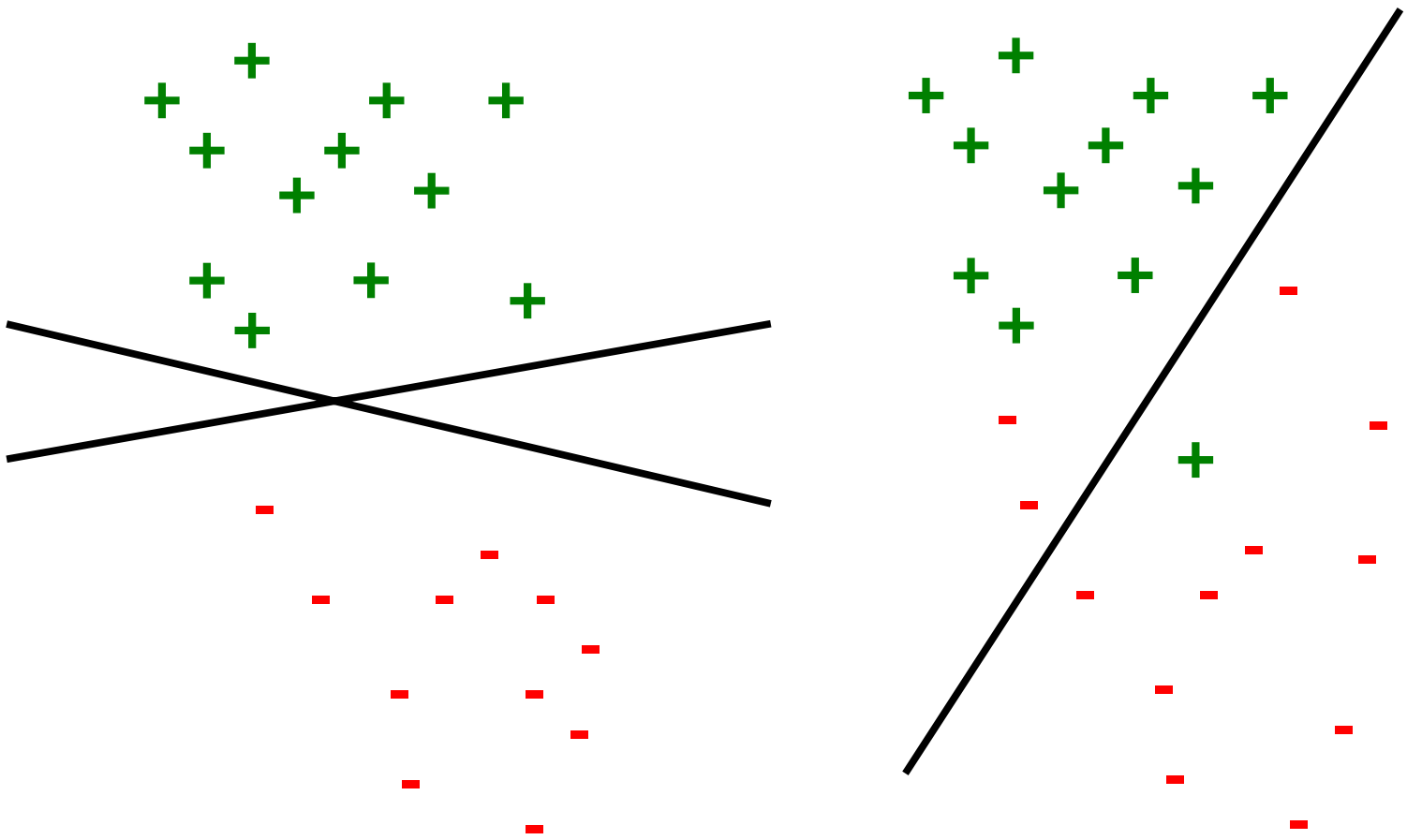
What kind of solution?

- Self-organizing local algorithms exist such as gossip, chaotic iteration, etc.
- These are
  - Are very fast and scalable
  - Are fairly simple to implement (but not always simple to analyze)
  - Provide almost exact, or often exact results in the face of very unreliable communication
- Our ambition is to achieve this for data mining algorithms (we focus on classification now)

Classification problem in machine learning

- We are given a set of (x_i,y_i) examples, where y_i is the class of x_i (y_i is eg. -1 or 1)
- We want a model f(), such that for all i, $f(x_i)=y$
- f() is very often a parameterized function $f_w()$, and the classification problem becomes an error minimization problem in w.
  - Neural net weights, linear model parameters, etc
- The error is often defined as a sum of errors over the examples

Illustration of classification with a linear model



Fully distributed classification

- So the problem is to find an optimization method that fits into our system and data model
- Most distributed methods build local models and then combine these through ensemble learning: but we don't have enough local data
- Online algorithms
  - Need only one data record at a time
  - They update the model using this record
- The stochastic gradient method is a popular online learning algorithm (we apply it to logistic regression)

Stochastic gradient descent

- Assume the error is defined as
- Then the gradient is
- So the full gradient method looks like
- But one can take only one example at a time iterating in random order over examples

$$Err(w) = \sum_{i=1}^{n} Err(w, x_i)$$

$$\frac{\partial Err(w)}{\partial w} = \sum_{i=1}^{n} \frac{\partial Err(w, x_i)}{\partial w}$$

$$w(t+1) = w(t) - \eta(t) \sum_{i=1}^{n} \frac{\partial Err(w, x_i)}{\partial w}$$

$$w(t+1) = w(t) - \eta(t) \frac{\partial Err(w, x_i)}{\partial w}$$

Gossip learning skeleton

Algorithm 1 Gossip Learning Framework

```
1:  c ← 0
2:  m ← initModel()
3:  currentModel ← initDriftHandler(m)
4:  receivedModels.add(currentModel)
5:  loop
6:      if receivedModels = ∅ then
7:          c ← c + 1
8:      if c = 10 then
9:          receivedModels.add(currentModel)
10:     for all m ∈ receivedModels do
11:         p ← selectPeer()
12:         send m to p
13:         receivedModels.remove(m)
14:         c ← 0
15:     wait(Δ)
16: procedure ONRECEIVEMODEL(m)
17:     m ← driftHandler(m)
18:     currentModel ← updateModel(m)
19:     receivedModels.add(currentModel)
```

Algorithm 2 Drift Handling Component

```
1:  procedure INITDRIFTHANDLER(m)
2:      m.TTL ← generateTTL()
3:      return m
4:  procedure DRIFTHANDLER(m)
5:      if m.age = m.TTL then
6:          m ← initModel()
7:          m ← initDriftHandler(m)
8:      return m
```

How to set the TTL

- We want to control the distribution of model ages at time t (A_t)
- But we can control only the distribution of TTL (X)
- The relationship between the two is given by

$$\mathbb{E}(A_t) \xrightarrow{t\to\infty} \frac{\mathbb{E}(X^2)}{2\mathbb{E}(X)}$$

$$\mathbb{D}^2(A_t) \xrightarrow{t\to\infty} \frac{\mathbb{E}(X^3)}{3\mathbb{E}(X)} - \left(\frac{\mathbb{E}(X^2)}{2\mathbb{E}(X)}\right)^2$$

Learning and Local prediction

- We use only the current local model for prediction
- The update rule is stochastic gradient for logistic regression

Algorithm 3 Learner Component

1: **procedure** INITMODEL
2: $m.age \leftarrow 0$
3: $m.w \leftarrow \bar{0}$
4: **return** m

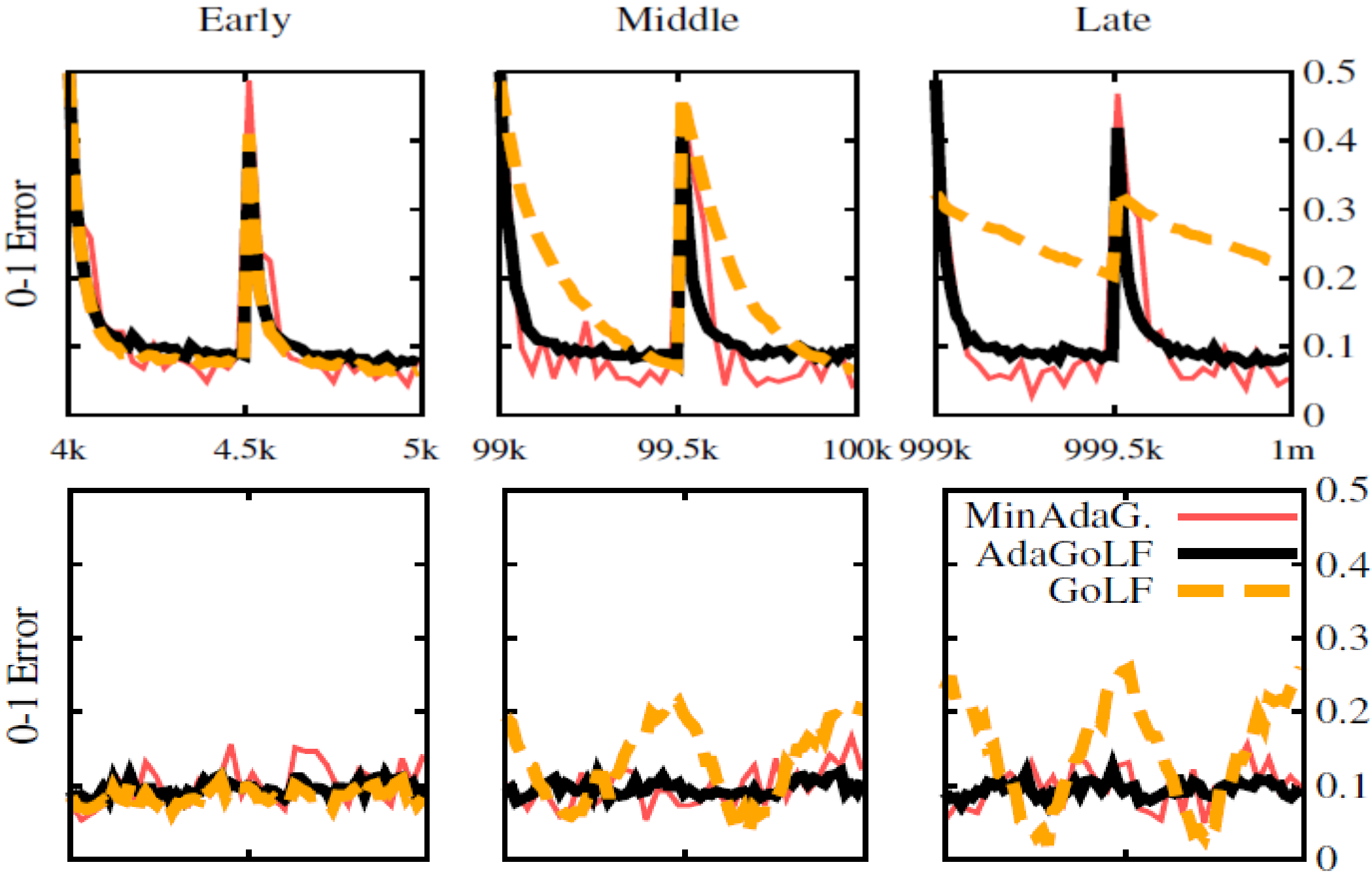
5: **procedure** UPDATEMODEL(m) ▷ $\lambda = 0.0001$
6: $m.age \leftarrow m.age + 1$
7: $m.\eta \leftarrow 1/m.age$
8: $\hat{y} \leftarrow m.\text{predict}(x)$ ▷ (x, y) is the local example
9: $m.w \leftarrow (1 - m.\eta \cdot \lambda)m.w + m.\eta \cdot (y - \hat{y})x$
10: **return** m

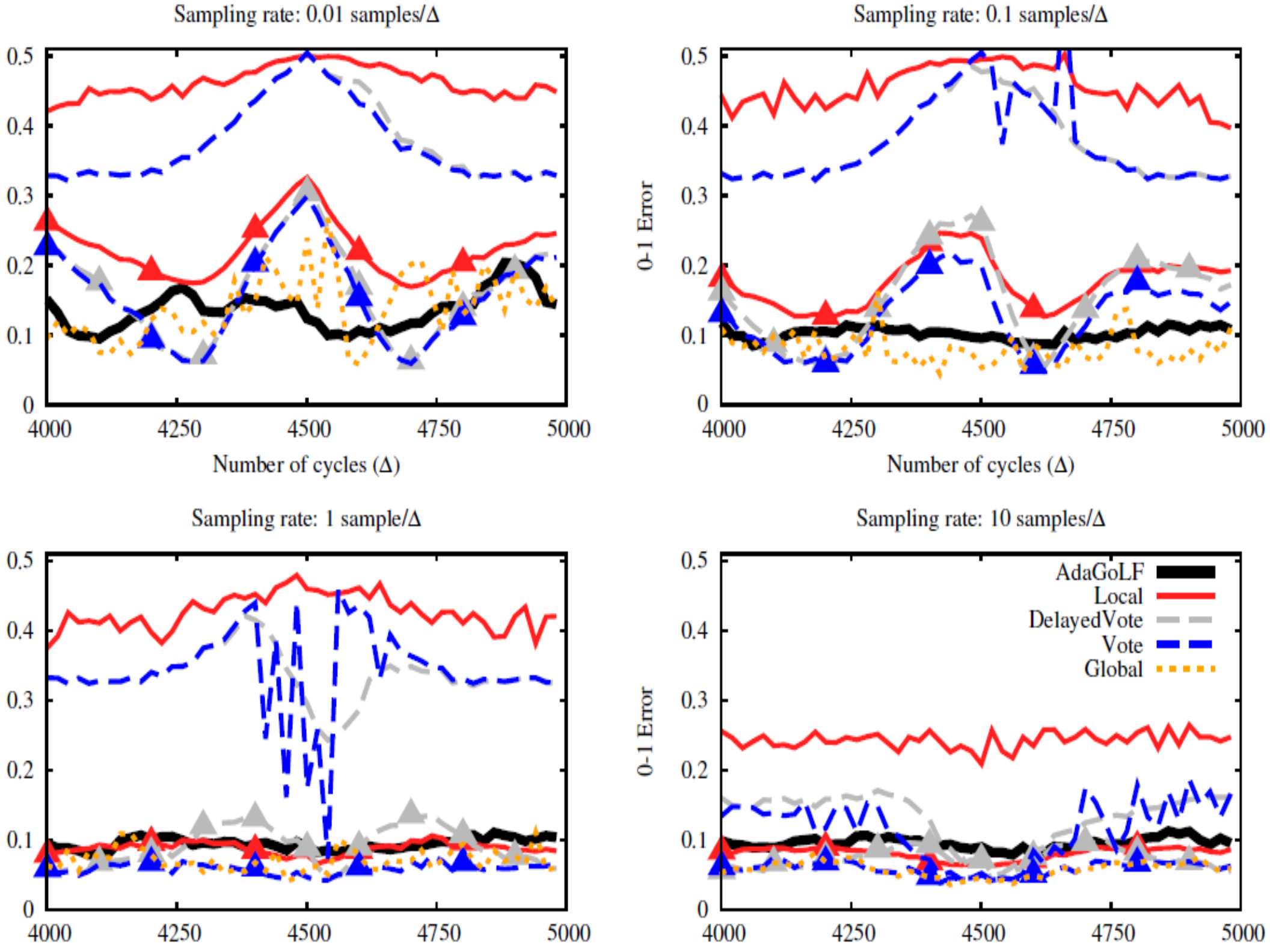
11: **procedure** PREDICT(x)
12: $p_0 \leftarrow 1/(1 + exp(\text{currentModel}.w^T x))$
13: $p_1 \leftarrow 1 - p_0$
14: **return** $p_0 > p_1$? 0 : 1

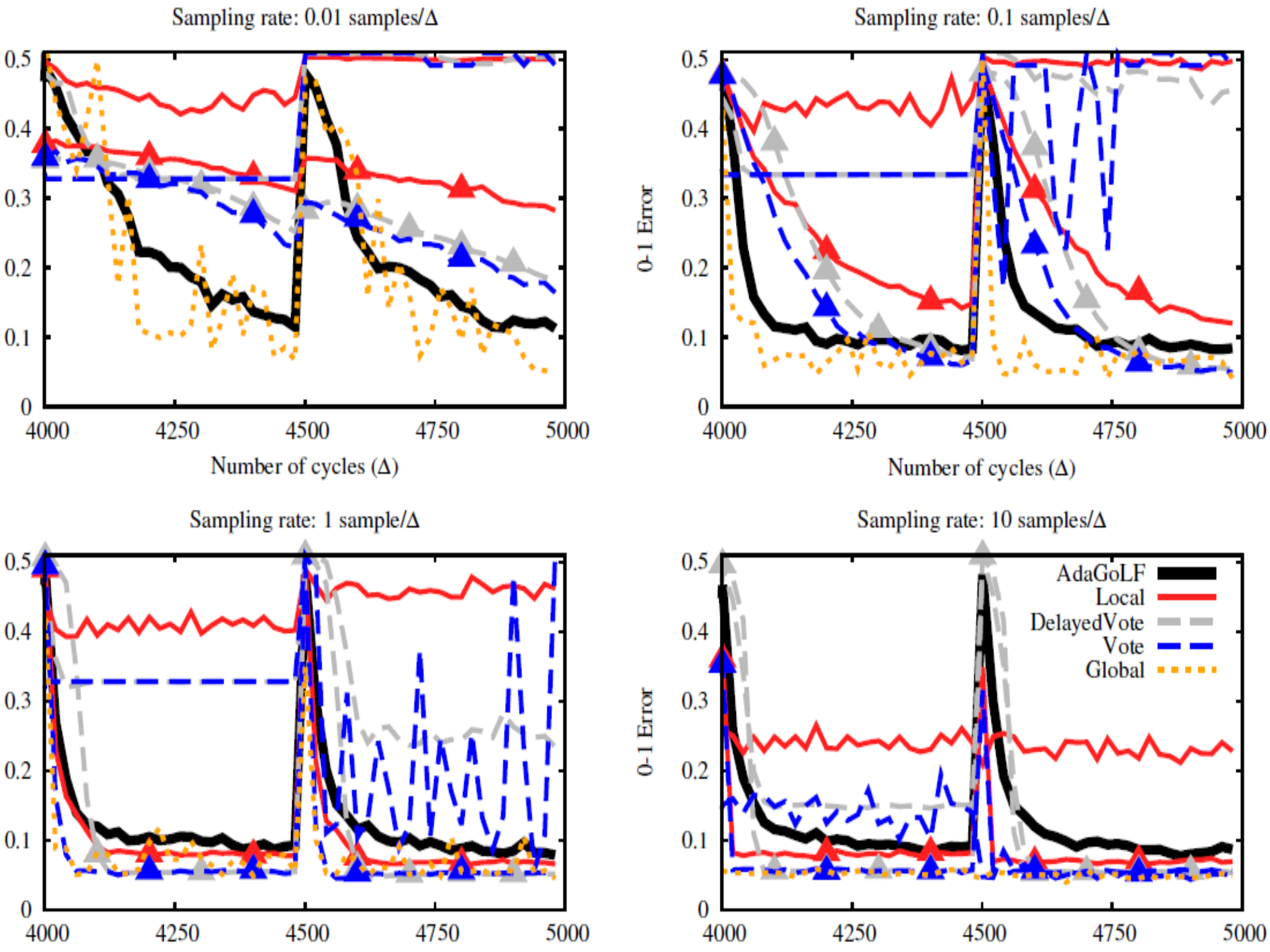
Experiments

- We used several benchmark drifting data sets for evaluations
  - Fully synthetic and synthetic labels on real data
  - Data is fully distributed: one data point per node
- We defined several baselines to cover solutions in related work
- We used extreme scenarios
  - 50% message drop rate
  - 1-10 cycles of message delay
  - Churn modeled after the FileList.org trace from Delft

The Drifting problem




Sampling rate: 0.01 samples/Δ
Sampling rate: 0.1 samples/Δ
Sampling rate: 1 sample/Δ
Sampling rate: 10 samples/Δ
0-1 Error
Number of cycles (Δ)
0
0.1
0.2
0.3
0.4
0.5
4000
4250
4500
4750
5000
AdaGoLF
Local
DelayedVote
Vote
Global


Sampling rate: 0.01 samples/Δ
Sampling rate: 0.1 samples/Δ
Sampling rate: 1 sample/Δ
Sampling rate: 10 samples/Δ
0-1 Error
Number of cycles (Δ)
0
0.1
0.2
0.3
0.4
0.5
4000
4250
4500
4750
5000
AdaGoLF
Local
DelayedVote
Vote
Global

Conclusions

- If the sampling rate is rare relative to drift speed, then our algorithm is favorable
  - Many improvements are possible, this is the “vanilla” version that uses only a single example in each cycle for update and uses no model merging
- Some results we did not discuss
  - Robustness to failure is good, a slowdown can be observed due to slower random walks
  - The algorithm is very insensitive to system size

Remarks regarding convergence

- If uniformity of random walk is guaranteed, then all the models converge to the true model eventually, irrespective of all failures
- If no uniformity can be guaranteed, but the local data is statistically independent of the visiting probability, then we will have no bias, but variance will increase (effectively we work with fewer samples)
- If no uniformity and no independence could be guaranteed, convergence to a good model is still ensured provided that the data is separable, and all misclassified examples are visited “often enough”

Publications

- Róbert Ormándi, István Hegedűs, and Márk Jelasity. Asynchronous peer-to-peer data mining with stochastic gradient descent. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, Euro-Par 2011, volume 6852 of Lecture Notes in Computer Science, pages 528–540. Springer-Verlag, 2011.
- Róbert Ormándi, István Hegedűs, and Márk Jelasity. Gossip learning with linear models on fully distributed data. Concurrency and Computation: Practice and Experience, 2012. to appear.
- István Hegedűs, Róbert Busa-Fekete, Róbert Ormándi, Márk Jelasity, and Balázs Kégl. Peer-to-peer multi-class boosting. In Euro-Par 2012, Lecture Notes in Computer Science. Springer-Verlag, 2012. to appear.