Asynchronous Peer-to-Peer Data Mining

Róbert Ormándi, István Hegedűs, and
Márk Jelasity

Motivation

- For P2P networks the only viable communication model is the chaotic model
  - Eventual update assumption, but
  - No reliability assumptions for individual messages
- In P2P networks one needs to build global models of distributed data for
  - Recommendations, spam filtering, optimizations of performance
- Does not hurt to have privacy preservation

P2P chaotic algorithms

- Examples of algorithms for which no provably chaotic version is known (not clear if possible at all)
  - Peer sampling
  - Average calculation
- Example of algorithms that do work in this model
  - Chaotic iteration (Lubachevsky and Mitra, 1986)
- We illustrate the problems through averaging

Illustration of averaging

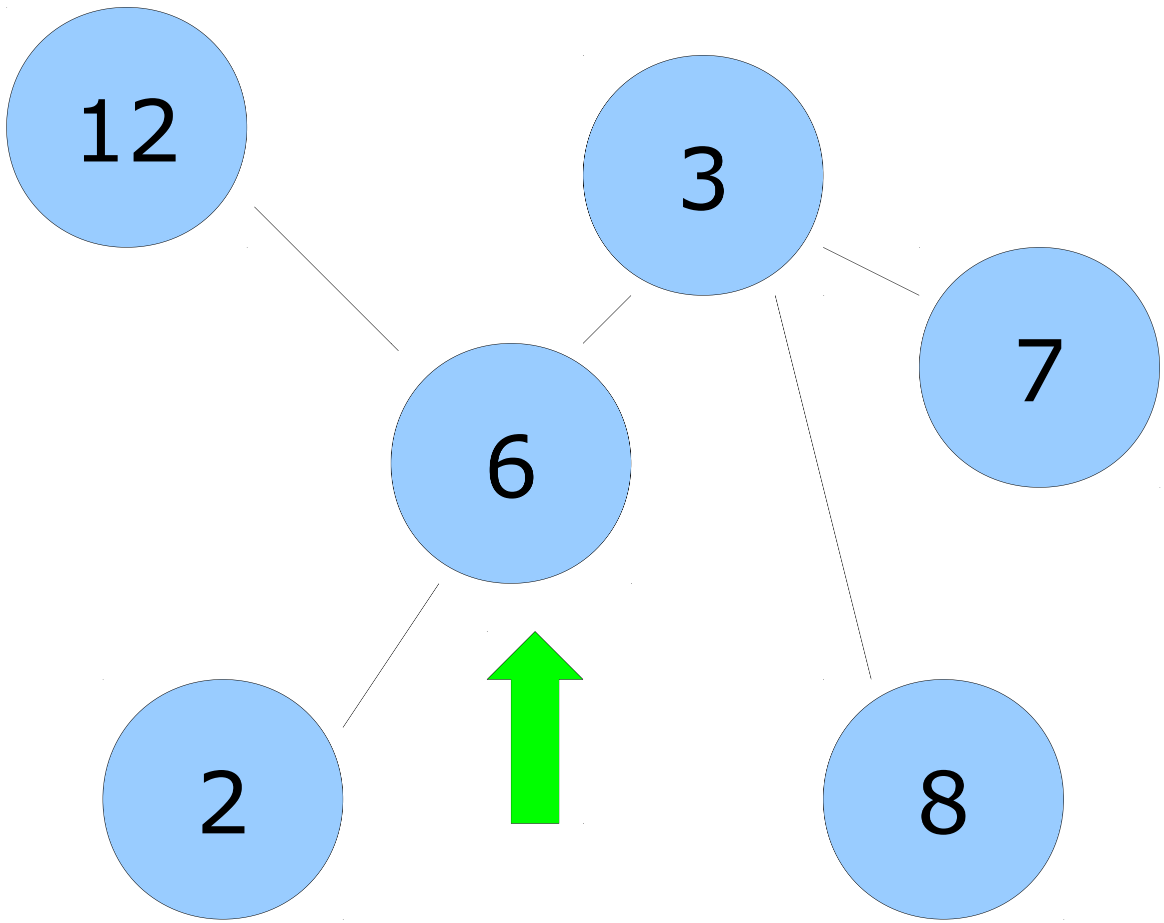


Illustration of averaging

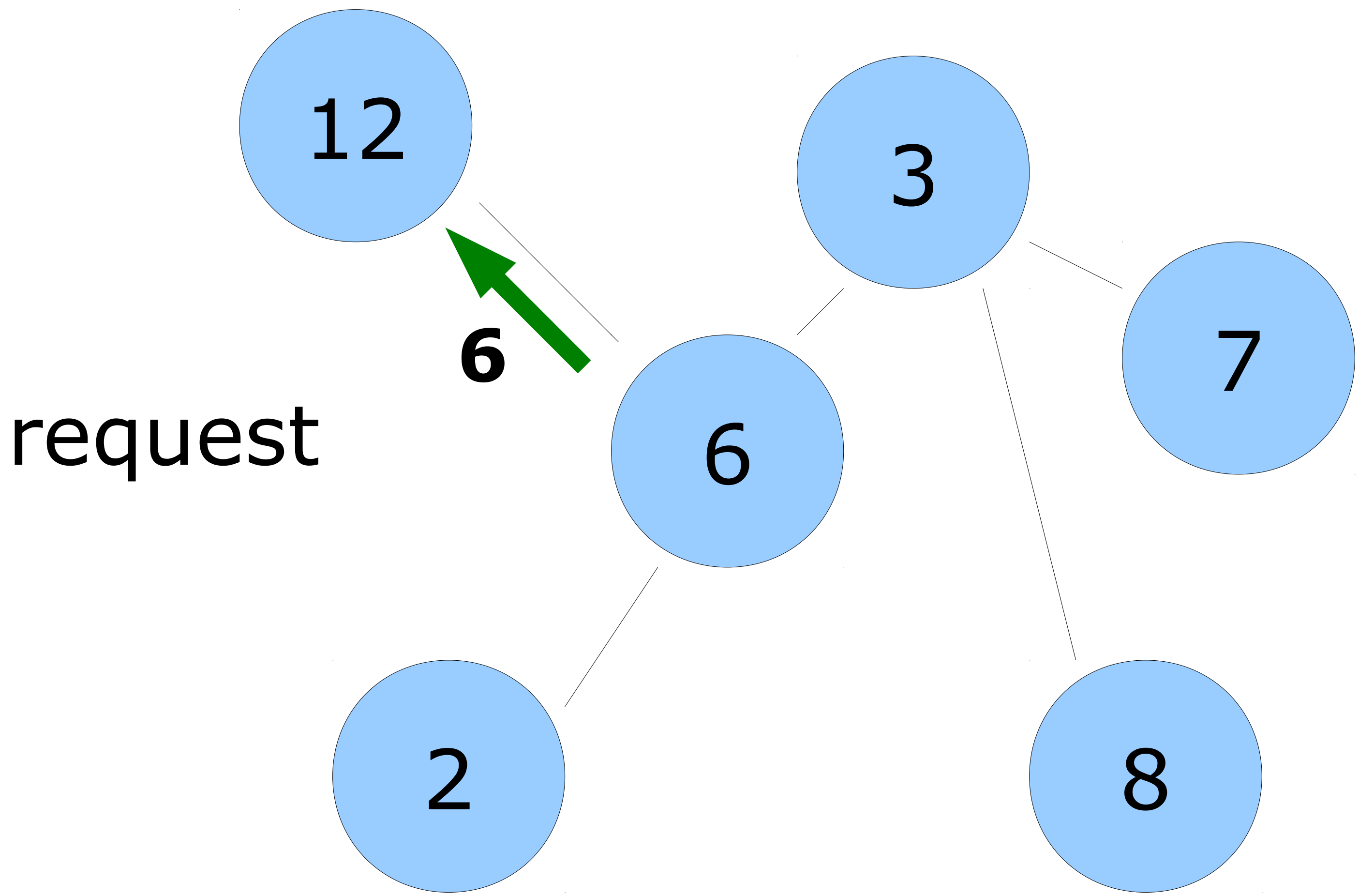
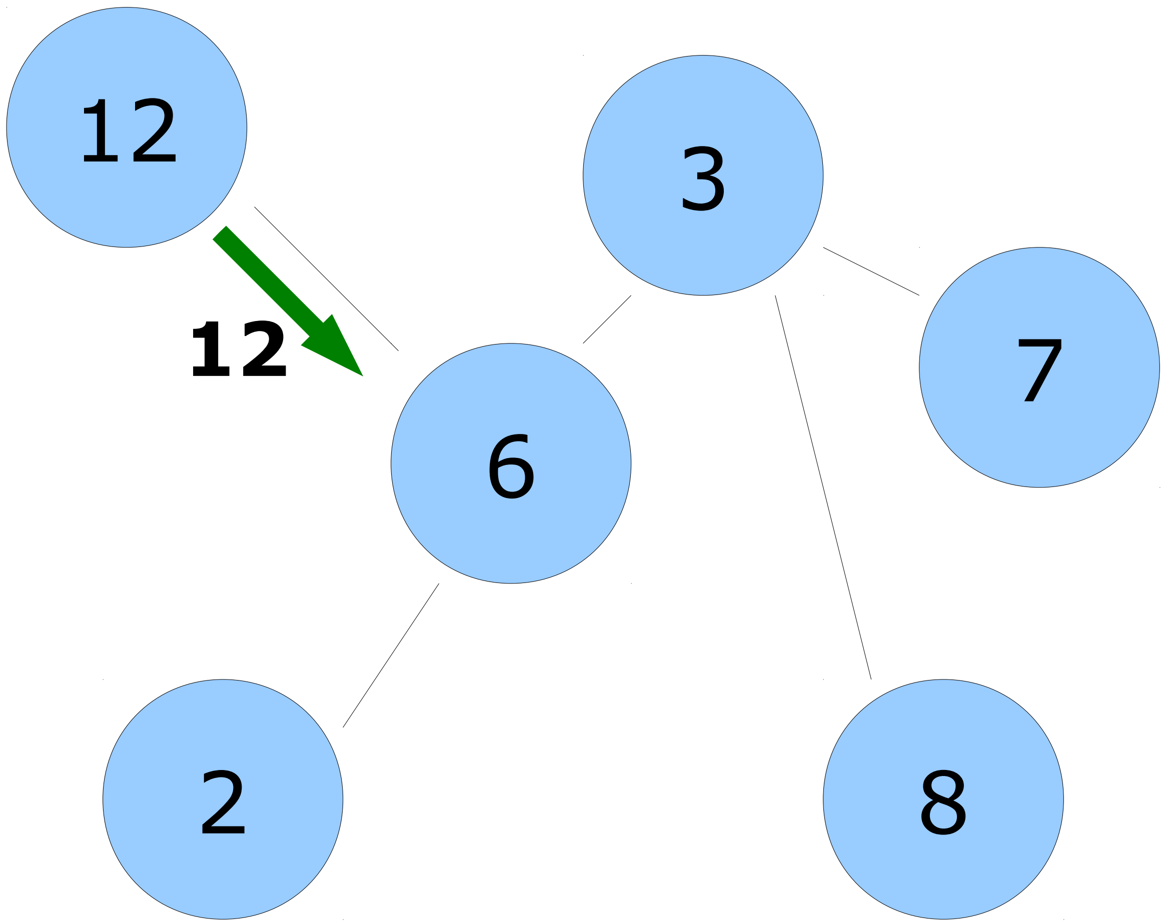
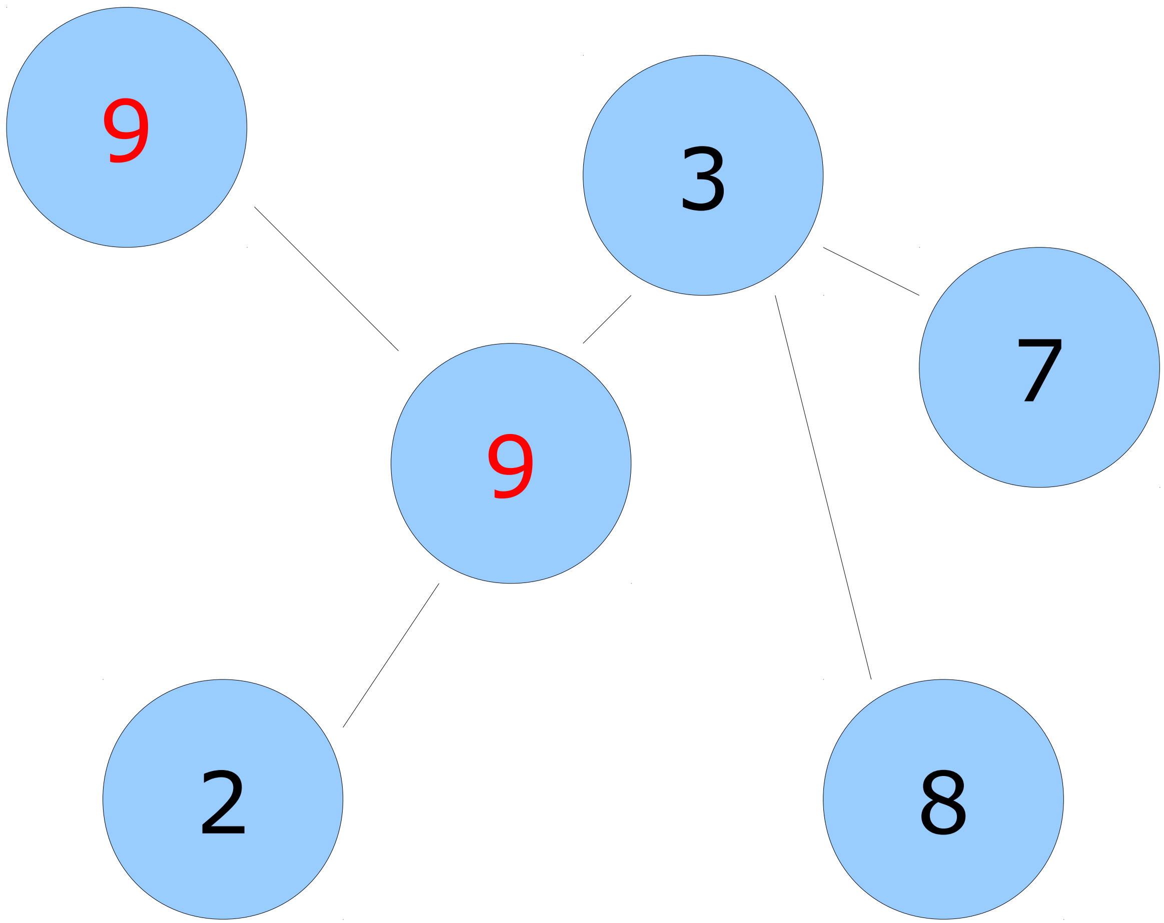


Illustration of averaging



reply

Illustration of averaging



(12+6)/2=9

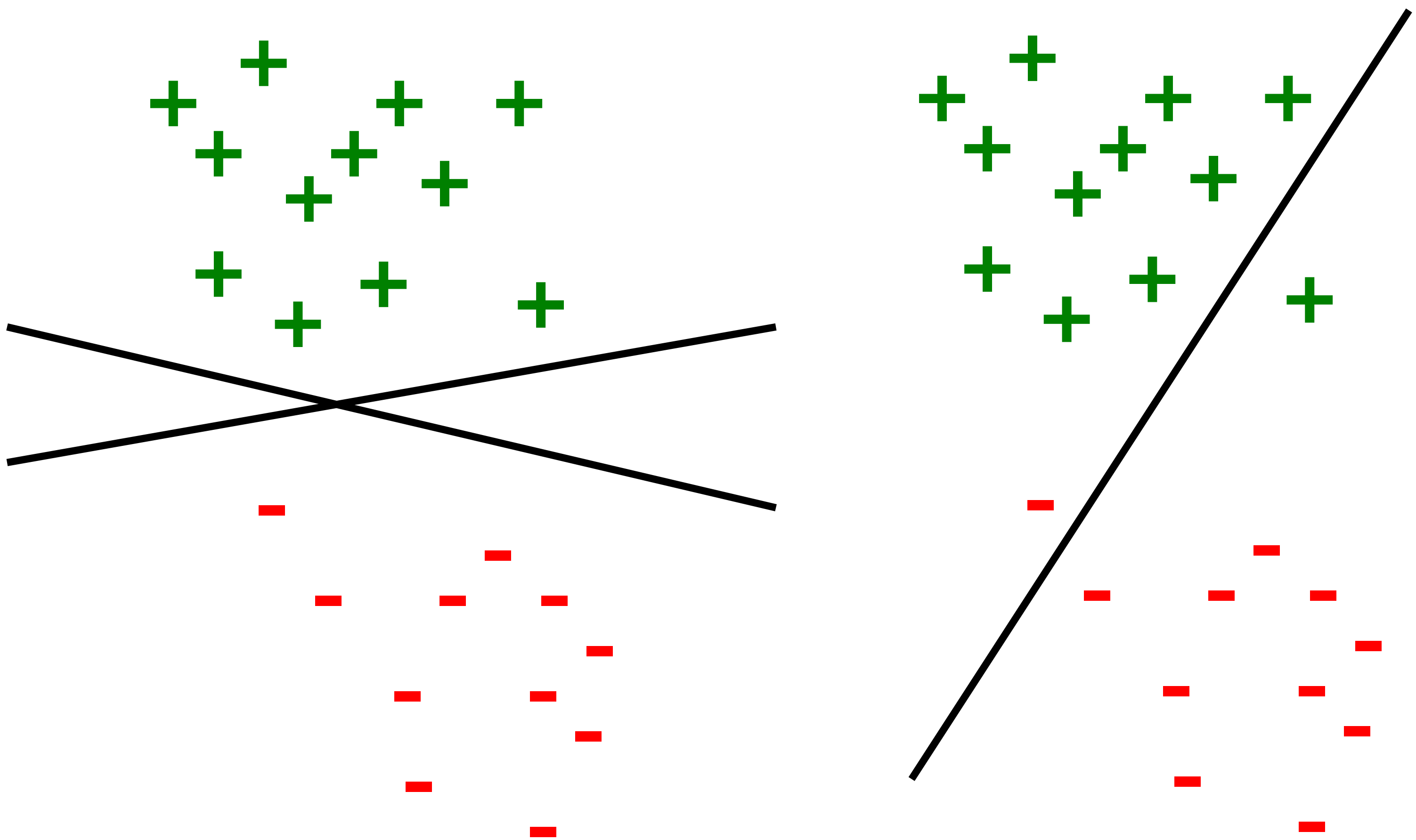
Problems

- Message delay: overlapping updates compromise the system state
  - Can be solved by blocking (deadlock can be avoided) or maintaining state for each link (Mehyar et al, 2007)
  - Can be also solved by the push-sum algorithm (this is the more elegant way) (Kempe et al, 2003)
- Message drop failures
  - No known solutions!
  - TCP is of course good enough in practice, but we do not want to rely on it in the chaotic model

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



Classification vs averaging

- Classification is more general
  - $f_w()$ can be in any form, and an optimal w can be any global function of the examples
- Classification is still easier
  - We do not always want a specific w, but instead we want one that classifies the examples well
  - Even if unique, an optimal w can depend on a small subset of examples only (borderline cases)

P2P classification

- So the problem is to solve the error minimization problem to find w (ie, train the model)
- For different model families (neural nets, support vector machines, etc) different sophisticated optimization algorithms are used most of which are very difficult to parallelize, let alone asynchronously
- Except stochastic gradient descent
  - Not always the most efficient but
  - A very generic method, very simple, and ideal for parallelization in our model

Stochastic gradient descent

- Assume the error is defined as

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

- Then the gradient is

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

- So the full gradient method looks like

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

- But one can take only one example at a time iterating in random order over examples

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

P2P stochastic gradient

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    m ← selectModel()
    send model(m) to p
end loop

procedure ONRECEIVEMODEL(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
end procedure
```

- SelectPeer is implemented by a peer sampling service for uniform random samples
- Depending on the implementation of updateModel we get different ML methods
- What can go wrong due to failures and delays is only the sampling distribution

P2P stochastic gradient

- Local prediction is based on locally available model(s)
- One can use the current model
- Or a combination of available models via eg voting

procedure PREDICT(x)
    $w \leftarrow currentModel$
    return sign($\langle w, x \rangle$)
end procedure

procedure VOTEDPREDICT(x)
    pRatio $\leftarrow 0$
    for $m \in$ modelQueue **do**
        if sign($\langle m.w, x \rangle$) ≥ 0 **then**
            pRatio $\leftarrow$ pRatio $+1$
        end if
    end for
    return sign(pRatio/modelQueue.size()-0.5)
end procedure

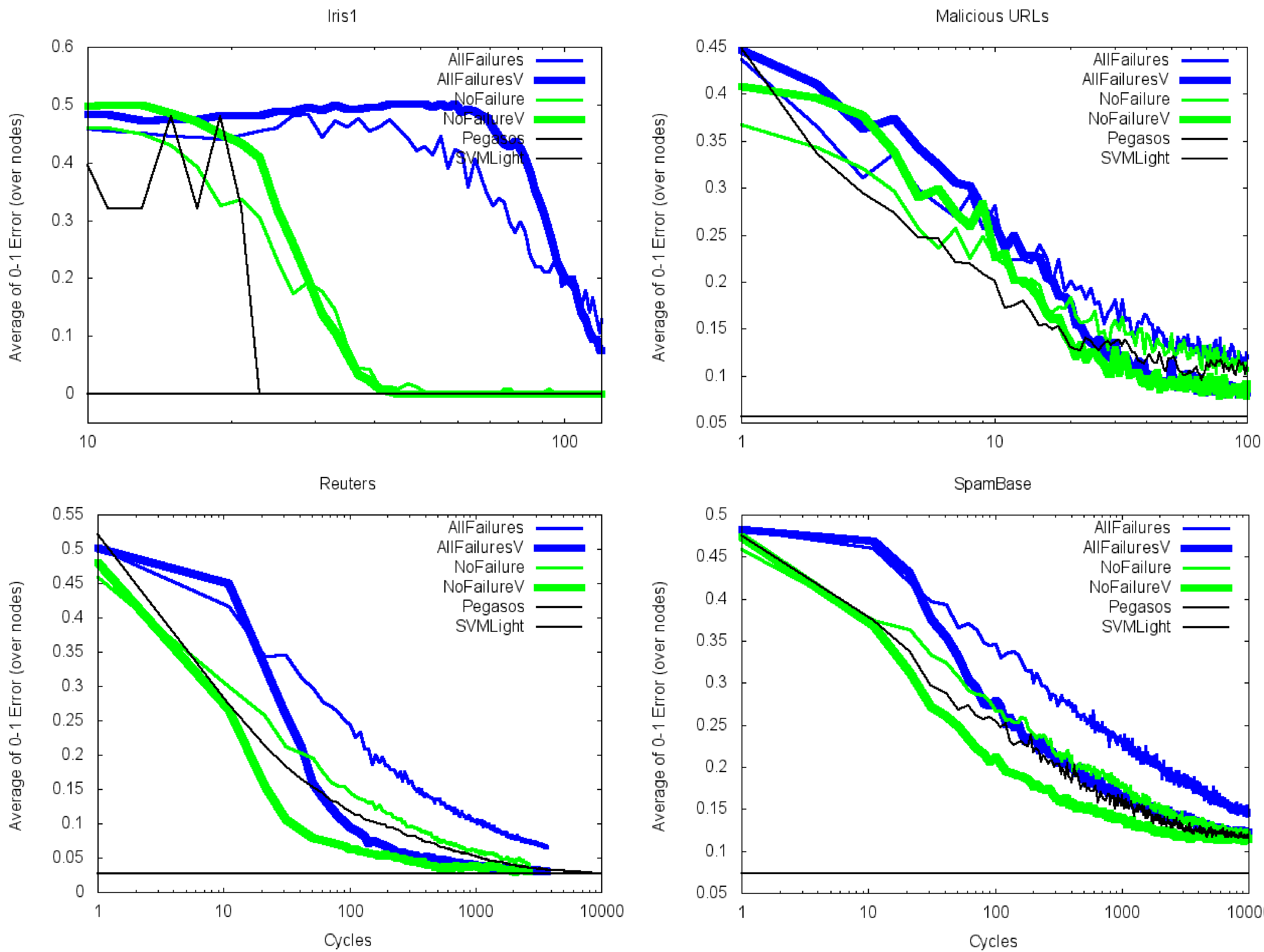
Experiments

- We implemented a support vector machine with stochastic gradient (Pegasos alg.)
- We used several benchmark data sets for evaluations
  - Data is fully distributed: one data point per node
- We used extreme scenarios
  - 50% message drop rate
  - 1-10 cycles of message delay
  - Churn modeled after the FileList.org trace from Delft

The data sets

	Iris1	Iris2	Irirs3	Reuters	SpamBase	Malicious10
Training set size	90	90	90	2000	4140	2155622
Test set size	10	10	10	600	461	240508
Number of features	4	4	4	9947	57	10
Classlabel ratio	50/50	50/50	50/50	1300/1300	1813/2788	792145/1603985
Pegasos 20000 iter.	0	0	0	0.025	0.111	0.080 (0.081)
Pegasos 1000 iter.	0	0	0.4	0.057	0.137	0.095 (0.060)
SVMLight	0	0	0.1	0.027	0.074	0.056 (–)

- Properties of data sets in the experiments
- Prediction error of baseline algorithms



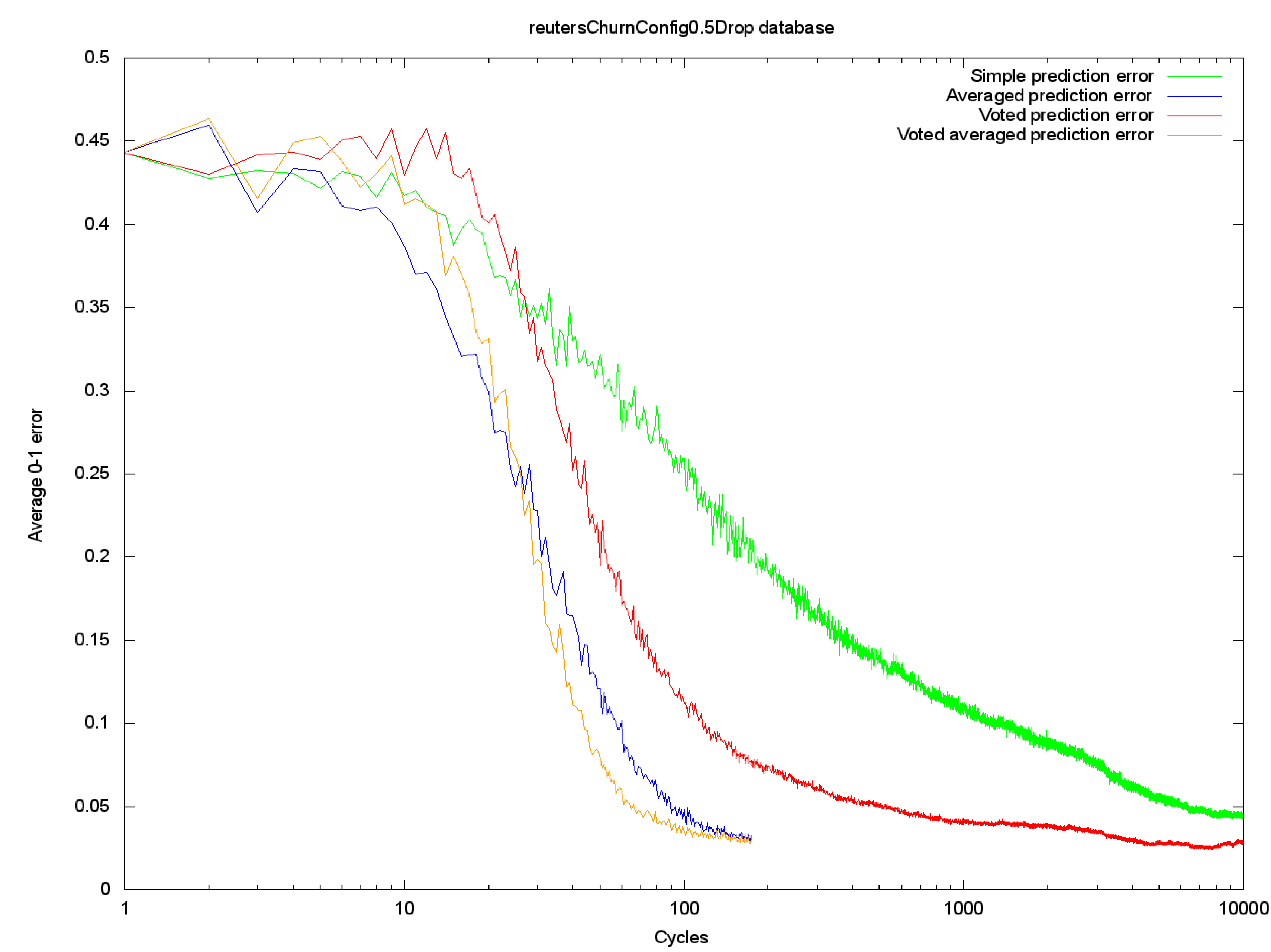
Iris1
Malicious URLs
Reuters
SpamBase
AllFailures
AllFailuresV
NoFailure
NoFailureV
Pegasos
SVMLight
Average of 0-1 Error (over nodes)
Cycles

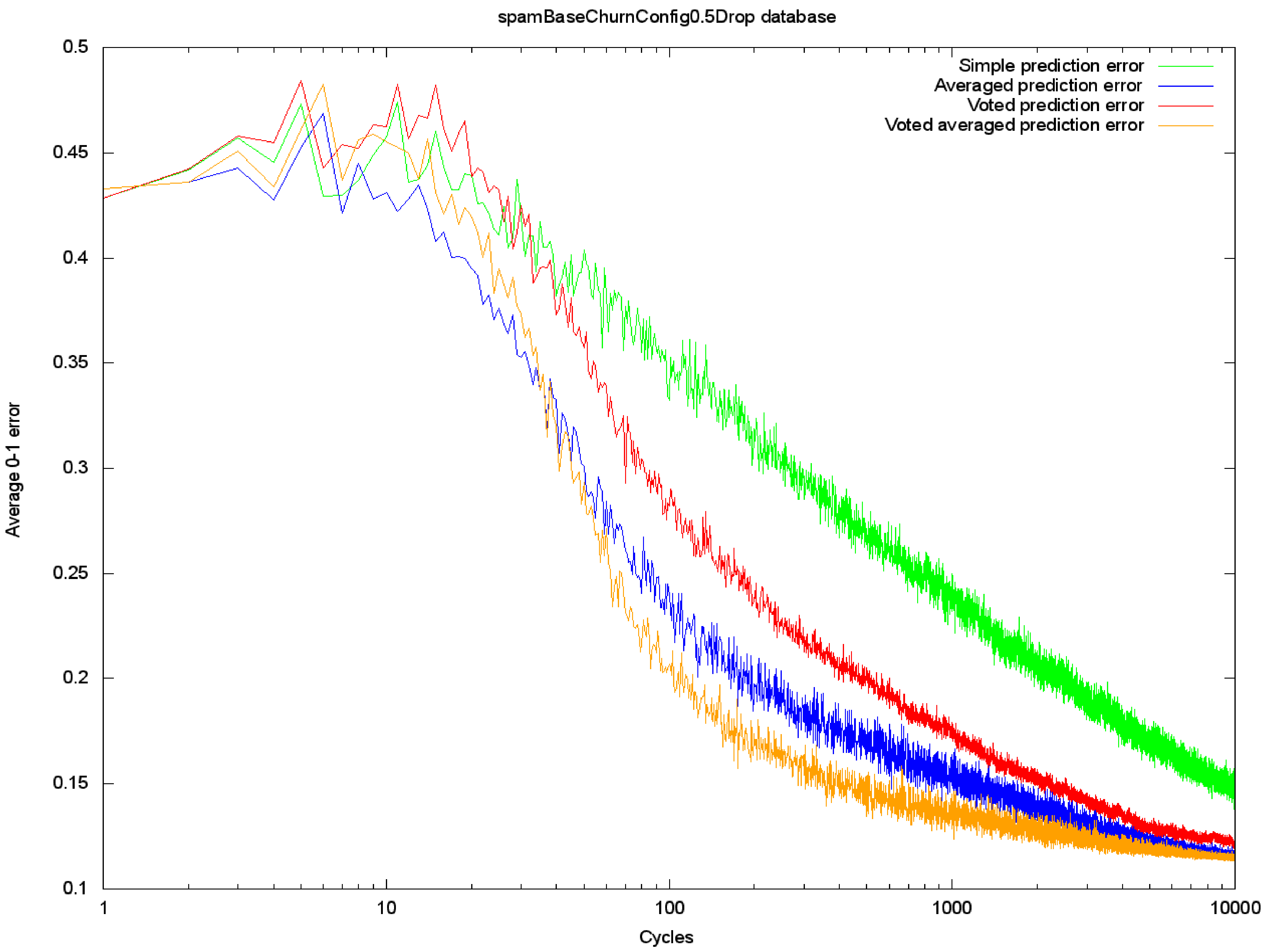
Remarks regarding the chaotic model

- 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 again convergence to the true model is guaranteed in general for all models
- 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”

Work in progress!

- Privacy preservation
  - Promising candidate, because the own value is revealed only in the first step at the moment
- Boosting efficiency
  - Models can be based on an exponential number of examples if we also average models at nodes (and not only update them); this seems to work...
- Dynamic fine-tuning
  - Based on local information the voting procedure, starting off new models, and deleting old ones could be controlled smartly


reutersChurnConfig0.5Drop database
Simple prediction error
Averaged prediction error
Voted prediction error
Voted averaged prediction error
Average 0-1 error
Cycles
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
10
100
1000
10000


spamBaseChurnConfig0.5Drop database
Simple prediction error
Averaged prediction error
Voted prediction error
Voted averaged prediction error
Average 0-1 error
Cycles
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
1
10
100
1000
10000