Asynchronous Peer-to-peer Data Mining with Stochastic Gradient Descent

Róbert Ormándi, István Hegedűs
and Márk Jelasity

EuroPar-2011

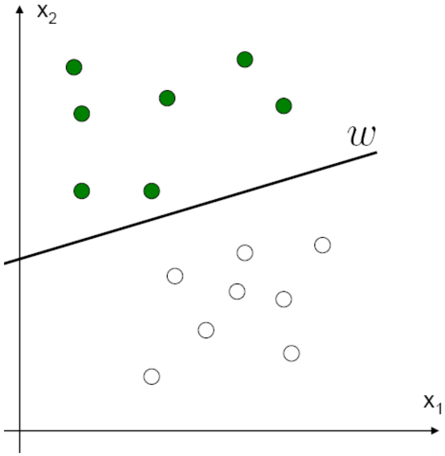
Motivation

- There are P2P algorithms that compute statistics on a distributed dataset
  - Min, max, avg …
- Our goal: using sophisticated machine learning (ML) algorithms in a P2P way
  - For supporting
    - spam filtering
    - opinion mining
    - personalized rank, search and recommendation
  - That is simple, cheap and robust
    - We use gossip-based techniques

Our System and Data Model

- Given a network of computers (nodes)
- The database is distributed in the network
  - Every node has exactly one training sample → training set size = network size
- Every node can get the address of a randomly selected node from the network
  - using the NewsCast peer sampling service
- Every node can send messages to another node if its address is available
- Every node should use locally the computed model

Classification



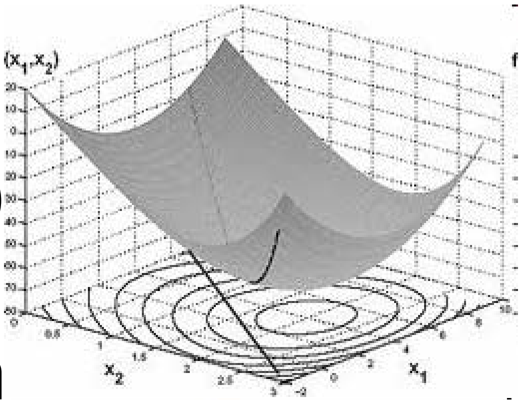
- Binary classification
  - Given $(x_1, y_1), \ldots, (x_n, y_n)$ training samples, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$
  - Task: looking for a model $f : \mathbb{R}^d \rightarrow \{-1, 1\}$ that correctly separates the samples from different classes →minimization problem of the formula

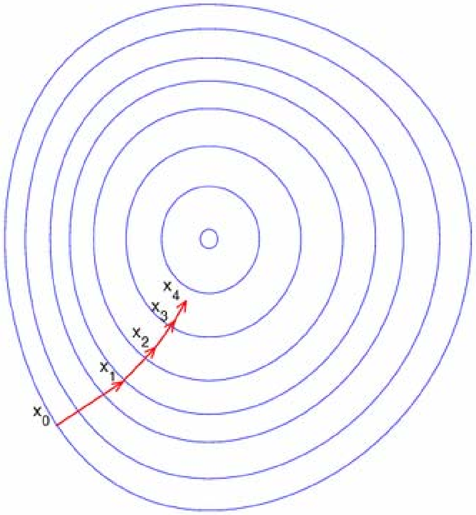
$$\min_f \sum_i (f(x_i) - y_i)^2 \quad i = 1, \ldots, n$$

  - In the linear case the model is a d dimensional hyperplane (w)
- Stochastic Gradient Descent is such a kind of method, that can find this model

Stochastic Gradient Descent (SGD) centralized case

- The SGD is an optimization algorithm for finding the minimum of an objective function ($(f(x_i) - y_i)^2$) using gradient steps
- Iteratively chooses a single uniform random sample from the training set for updating the model based on the gradient step
- Why SGD: Uses only one training sample at a time instead of the whole training set





Stochastic Gradient Descent (SGD) centralized case

- 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 gradient update is

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

- But SGD makes update based on only one sample

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

P2P SGD

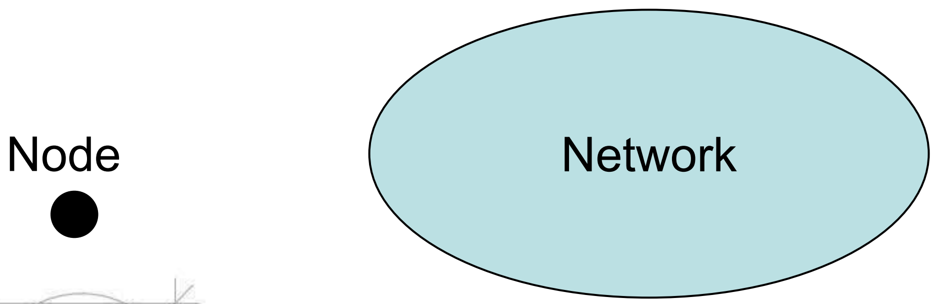
- Basic idea: models jump from node to node and they update themselves based on the local training sample
- We have the same SGD algorithm in a turned up manner
  - if we can guarantee the uniformity of peer sampling then our SGD models will converge to the optimum
- Privacy aspect: the data never leaves the node, just the model

Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

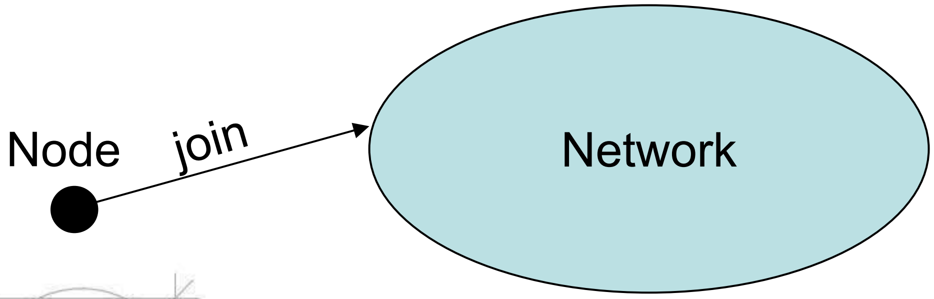


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

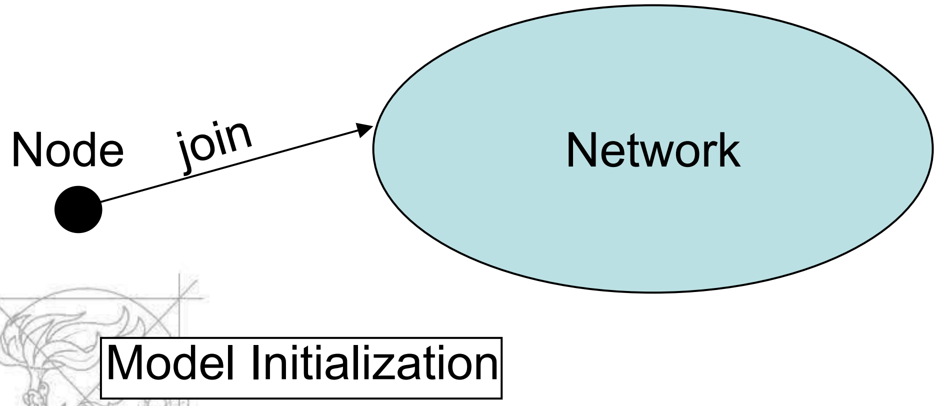


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

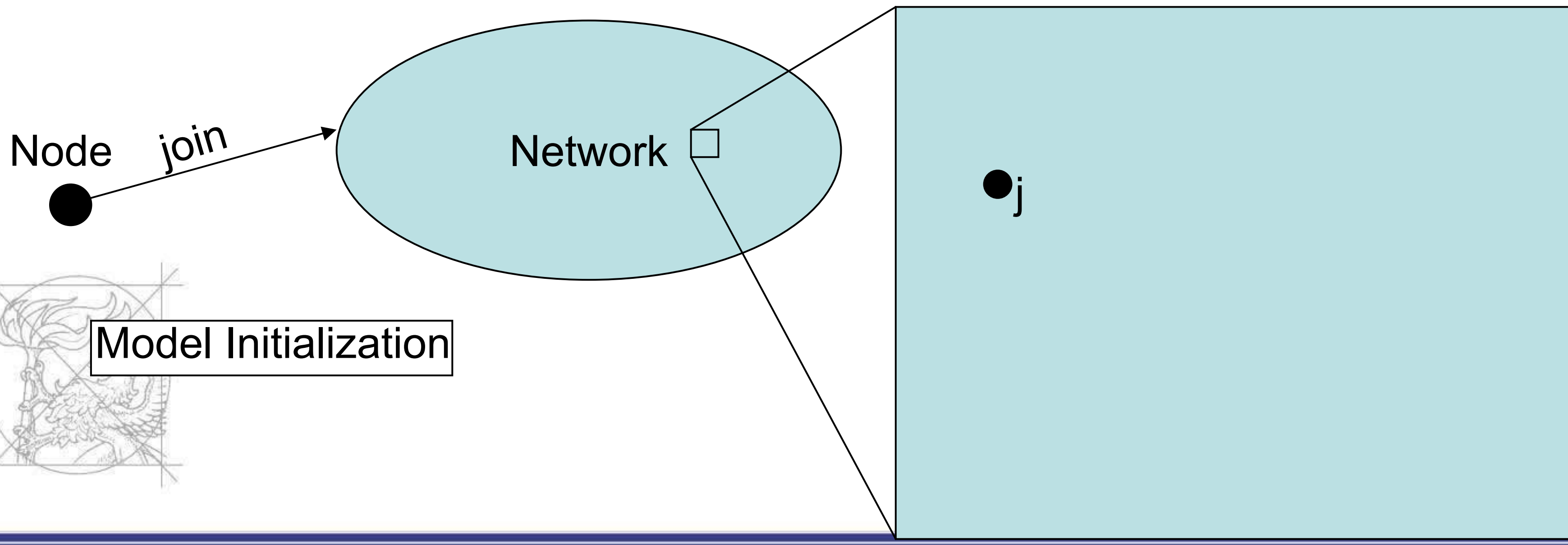


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

initModel()
loop
    wait(Δ)
    $p \leftarrow$ selectPeer()
    send currentModel to p

procedure ONRECEIVEMODEL(m)
    $m \leftarrow$ updateModel(m)
    currentModel $\leftarrow m$
    modelQueue.add(m)

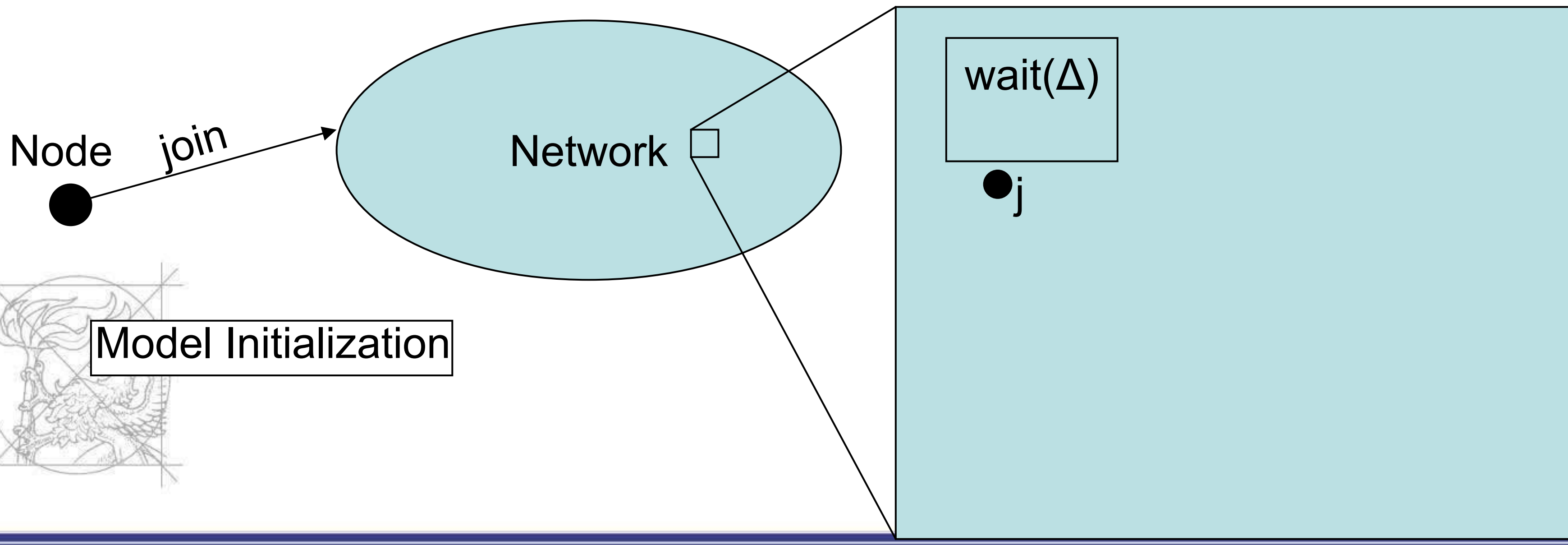


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

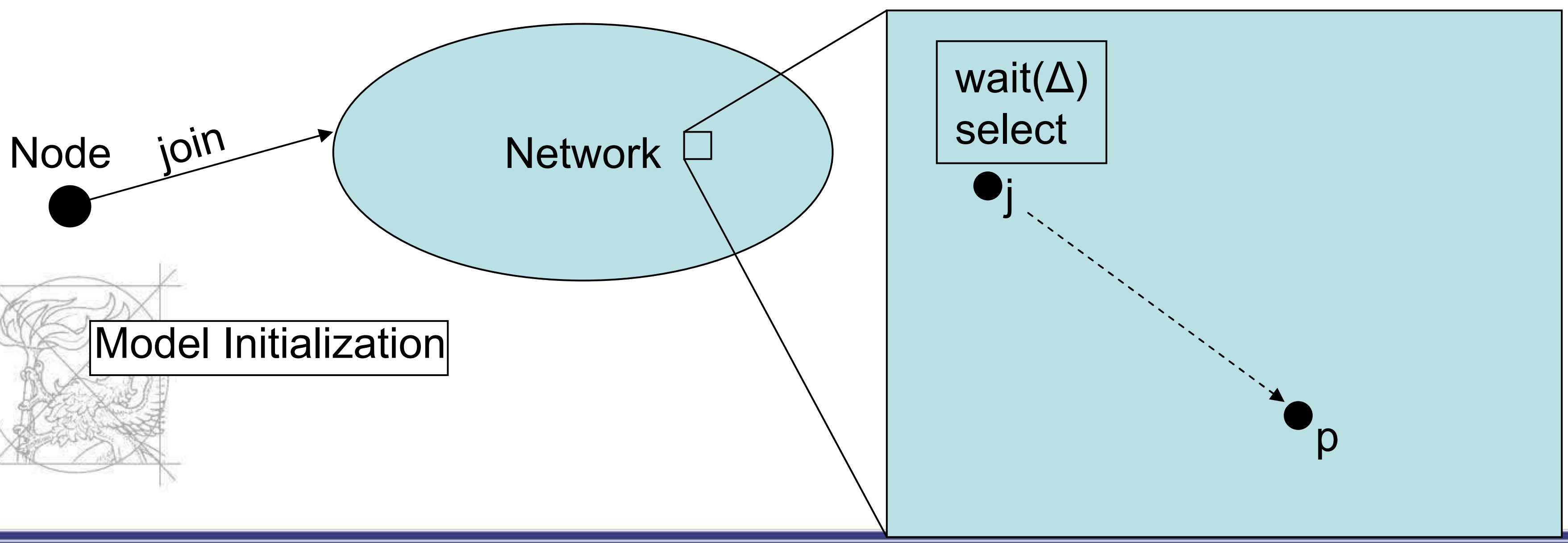


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

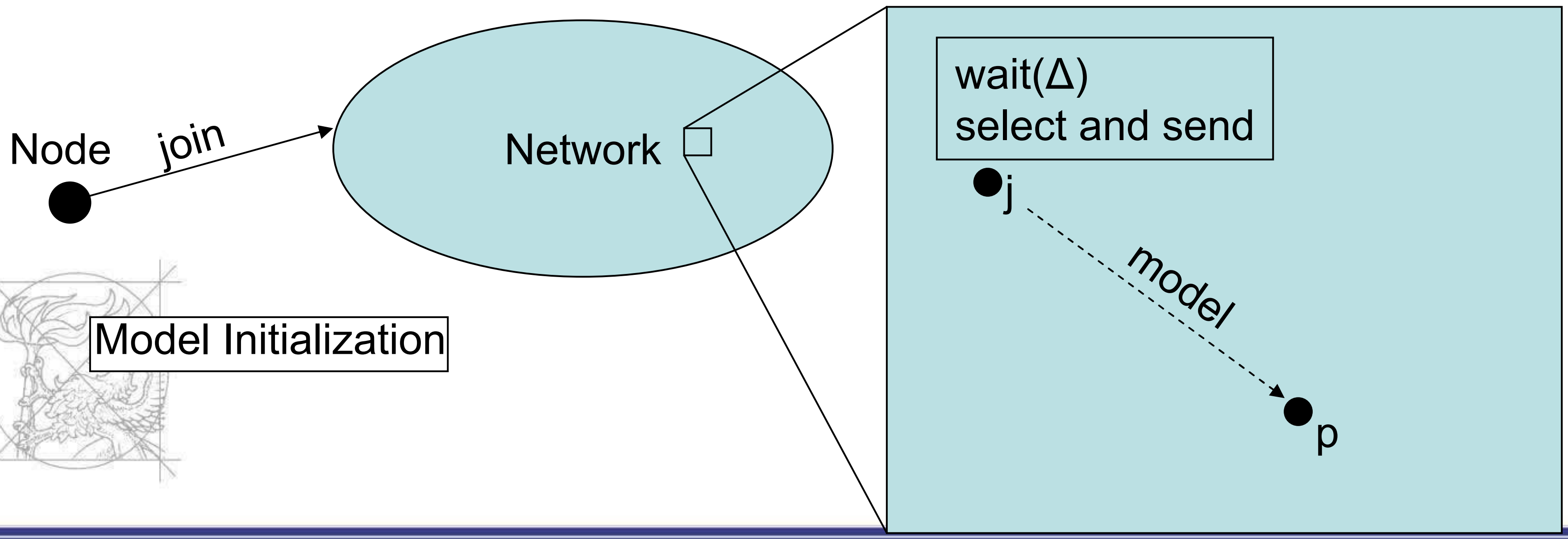


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

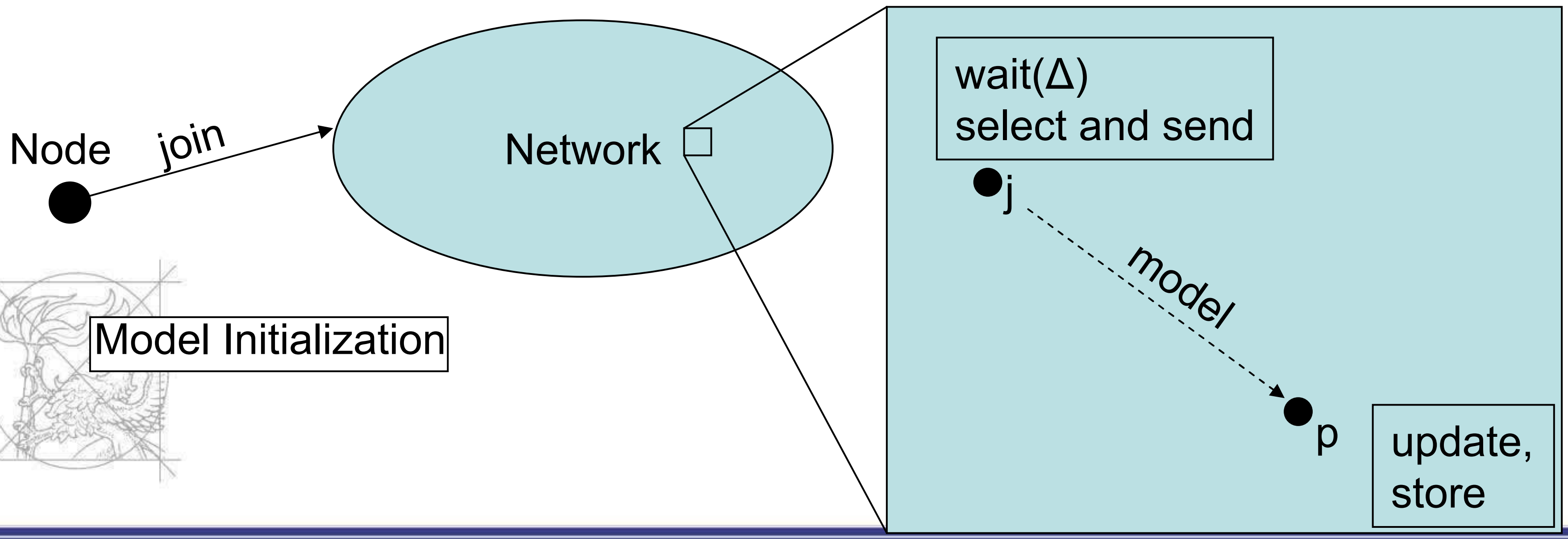


Our Data Mining Framework

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

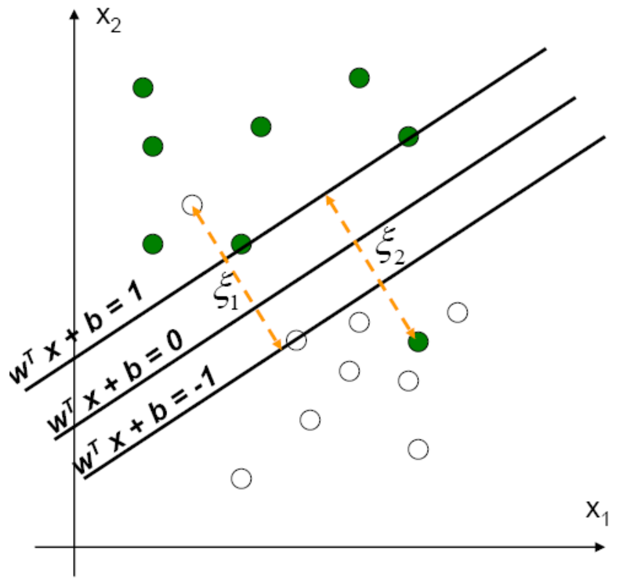
```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

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



Support Vector Machines (SVM) through SGD

- Pegasos SVM
  - SGD based ML algorithm
  - It is looking for a separating hyperplane (w) that separates examples of the two classes and maximizes the margin



$$\min_{w,b,\xi_i} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \quad (\forall i:\ 1 \leq i \leq n)$$

P2Pegasos SVM Algorithm

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

initModel()
loop
    wait(Δ)
    $p \leftarrow$ selectPeer()
    send currentModel to p

procedure ONRECEIVEMODEL(m)
    $m \leftarrow$ updateModel(m)
    currentModel $\leftarrow m$
    modelQueue.add(m)

- Define updateModel and initModel methods for different machine learning algorithms
- For Pegasos:

Algorithm 2 P2Pegasos

procedure UPDATEMODEL(m)
    $\eta \leftarrow 1/(\lambda \cdot m.t)$
    if $y \langle m.w, x \rangle < 1$ **then**
        $m.w \leftarrow (1 - \eta\lambda) m.w + \eta y x$
    else
        $m.w \leftarrow (1 - \eta\lambda) m.w$
    $m.t \leftarrow m.t + 1$
    return m

procedure INITMODEL
    $m.t \leftarrow 0$
    $m.w \leftarrow (0, \ldots, 0)^T$
    send model(m) to self

P2Pegasos SVM Algorithm

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

initModel()
loop
    wait(Δ)
    $p \leftarrow$ selectPeer()
    send currentModel to p

procedure ONRECEIVEMODEL(m)
    $m \leftarrow$ updateModel(m)
    currentModel $\leftarrow m$
    modelQueue.add(m)

- Define updateModel and initModel methods for different machine learning algorithms
- For Pegasos:

Algorithm 2 P2Pegasos

procedure UPDATEMODEL(m)
    $\eta \leftarrow 1/(\lambda \cdot m.t)$
    if $y \langle m.w, x \rangle < 1$ **then**
        $m.w \leftarrow (1 - \eta\lambda)m.w + \eta y x$
    else
        $m.w \leftarrow (1 - \eta\lambda)m.w$
    $m.t \leftarrow m.t + 1$
    return m

procedure INITMODEL
    $m.t \leftarrow 0$
    $m.w \leftarrow (0, \ldots, 0)^T$
    send model(m) to self

P2Pegasos SVM Algorithm

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
initModel()
loop
    wait(Δ)
    p ← selectPeer()
    send currentModel to p

procedure onReceiveModel(m)
    m ← updateModel(m)
    currentModel ← m
    modelQueue.add(m)
```

- Define updateModel and initModel methods for different machine learning algorithms
- For Pegasos:

Algorithm 2 P2Pegasos

```
procedure updateModel(m)
    η ← 1/(λ · m.t)
    if y ⟨m.w, x⟩ < 1 then
        m.w ← (1 − ηλ)m.w + ηyx
    else
        m.w ← (1 − ηλ)m.w
    m.t ← m.t + 1
    return m

procedure initModel
    m.t ← 0
    m.w ← (0, . . . , 0)^T
    send model(m) to self
```

P2Pegasos SVM Algorithm

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

initModel()
loop
    wait(Δ)
    $p \leftarrow$ selectPeer()
    send currentModel to p

procedure ONRECEIVEMODEL(m)
    $m \leftarrow$ updateModel(m)
    currentModel $\leftarrow m$
    modelQueue.add(m)

- Define updateModel and initModel methods for different machine learning algorithms
- For Pegasos:

Algorithm 2 P2Pegasos

procedure UPDATEMODEL(m)
    $\eta \leftarrow 1/(\lambda \cdot m.t)$
    if $y \langle m.w, x \rangle < 1$ **then**
        $m.w \leftarrow (1 - \eta\lambda)m.w + \eta y x$
    else
        $m.w \leftarrow (1 - \eta\lambda)m.w$
    $m.t \leftarrow m.t + 1$
    return m

procedure INITMODEL
    $m.t \leftarrow 0$
    $m.w \leftarrow (0, \ldots, 0)^T$
    send model(m) to self

P2Pegasos SVM Algorithm

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

initModel()
loop
  wait(Δ)
  $p \leftarrow$ selectPeer()
  send currentModel to p

procedure ONRECEIVEMODEL(m)
  $m \leftarrow$ updateModel(m)
  currentModel $\leftarrow m$
  modelQueue.add(m)

- Define updateModel and initModel methods for different machine learning algorithms
- For Pegasos:

Algorithm 2 P2Pegasos

procedure UPDATEMODEL(m)
  $\eta \leftarrow 1/(\lambda \cdot m.t)$
  if $y \langle m.w, x \rangle < 1$ **then**
    $m.w \leftarrow (1 - \eta\lambda)m.w + \eta y x$
  else
    $m.w \leftarrow (1 - \eta\lambda)m.w$
  $m.t \leftarrow m.t + 1$
  return m

procedure INITMODEL
  $m.t \leftarrow 0$
  $m.w \leftarrow (0, \ldots, 0)^T$
  send model(m) to self

P2Pegasos SVM Algorithm

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

initModel()
loop
    wait(Δ)
    $p \leftarrow$ selectPeer()
    send currentModel to p

procedure ONRECEIVEMODEL(m)
    $m \leftarrow$ updateModel(m)
    currentModel $\leftarrow m$
    modelQueue.add(m)

- Define updateModel and initModel methods for different machine learning algorithms
- For Pegasos:

Algorithm 2 P2Pegasos

procedure UPDATEMODEL(m)
    $\eta \leftarrow 1/(\lambda \cdot m.t)$
    if $y \langle m.w, x \rangle < 1$ **then**
        $m.w \leftarrow (1 - \eta\lambda)m.w + \eta y x$
    else
        $m.w \leftarrow (1 - \eta\lambda)m.w$
    $m.t \leftarrow m.t + 1$
    return m

procedure INITMODEL
    $m.t \leftarrow 0$
    $m.w \leftarrow (0, \ldots, 0)^T$
    send model(m) to self

Local Prediction

- Every node has a bounded queue of the last few received models for making prediction
  - Using a single model

```
procedure PREDICT(x)
    w ← currentModel
    return sign(⟨w, x⟩)
```

  - Using multiple models (majority voting)

```
procedure VOTEDPREDICT(x)
    pRatio ← 0
    for m ∈ modelQueue do
        if sign(⟨m.w, x⟩) ≥ 0 then
            pRatio ← pRatio +1
    return sign(pRatio/modelQueue.size()−0.5)
```

Communication Cost

- The cost of communication is exactly the same as the cost of other gossip based algorithms
  - Every node sends a message (model) to a uniform randomly selected node in each Δ time period
  - O(n) in the network
  - O(1) for a node
  - It includes the cost of peer sampling service as well

Algorithm Summary

- Random walk based P2P SGD method
  - Sending model instead of selecting sample
- Nodes collect model(s)
  - Simple and voted prediction
- Local prediction
  - No extra communication cost
- Each node runs the same algorithm
- Asynchronous

Experimental Setup

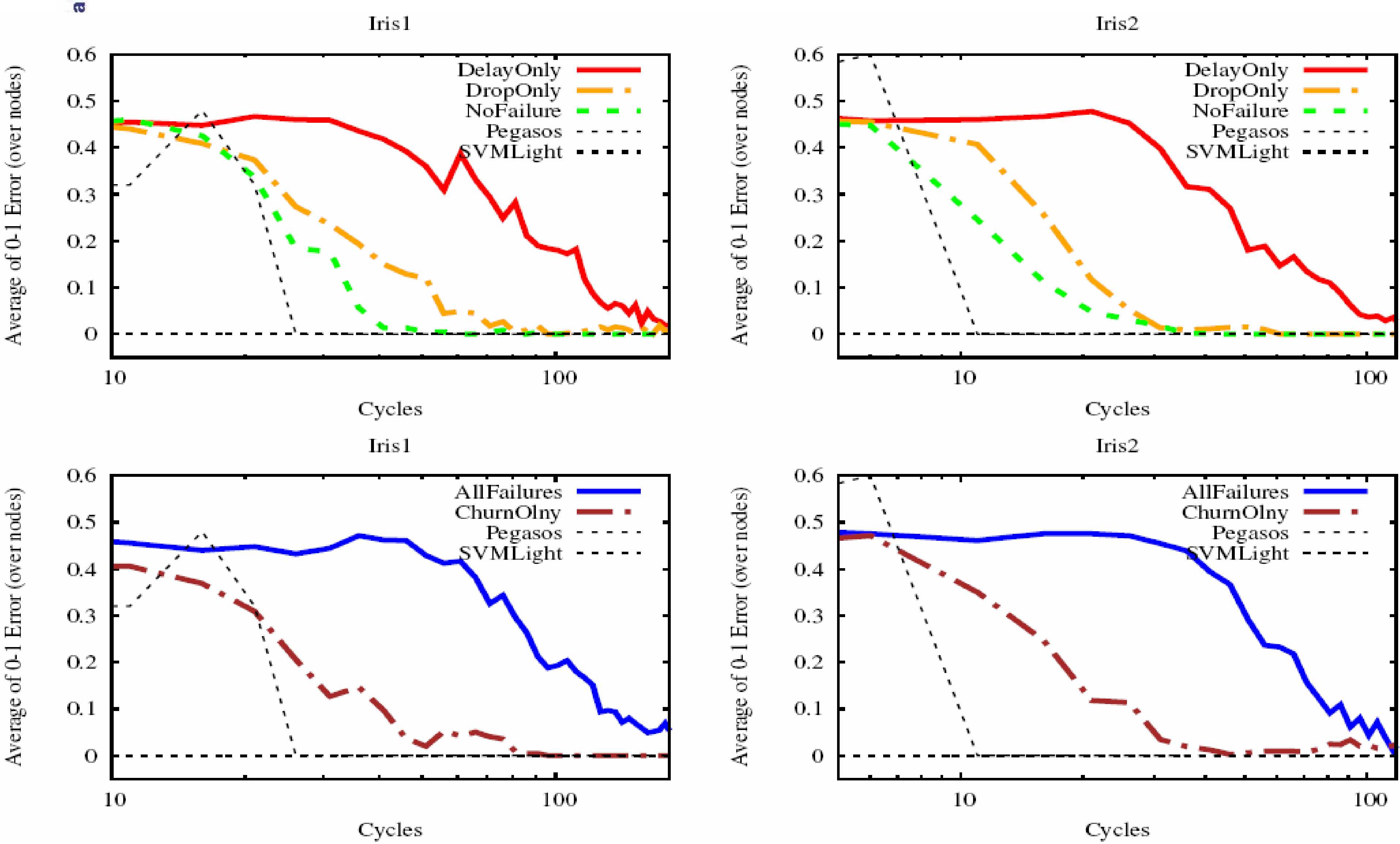
- PeerSim as a simulation environment
- NewsCast for peer sampling service
- The 10 latest models for Voting prediction
- Baselines: Pegasos, SVMLight
- Datasets: Iris, Reuters, SpamBase, Malicious
- Scenarios:
  - No failure
  - Drop only: 0.5 probability
  - Delay only: uniform random from [Δ, 10Δ]
  - Churn only: from approx. Log-normal dist.
  - All failures

Database Properties

	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 (–)

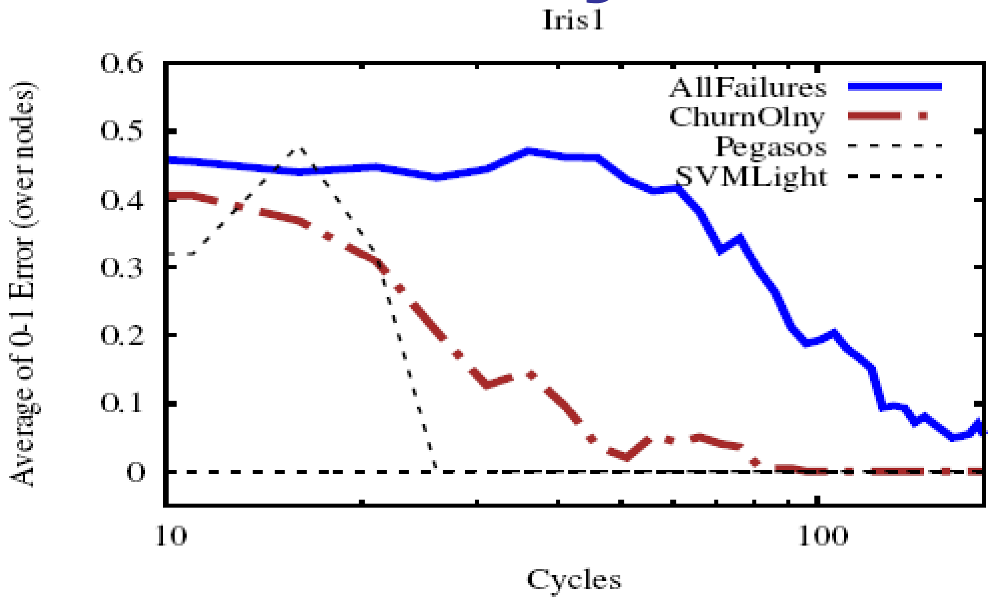
- Different types of datasets:
  - Small – large num. of samples
  - Small – large num. of features
- Split Iris and reduced Malicious features
- Performance of baseline algorithms

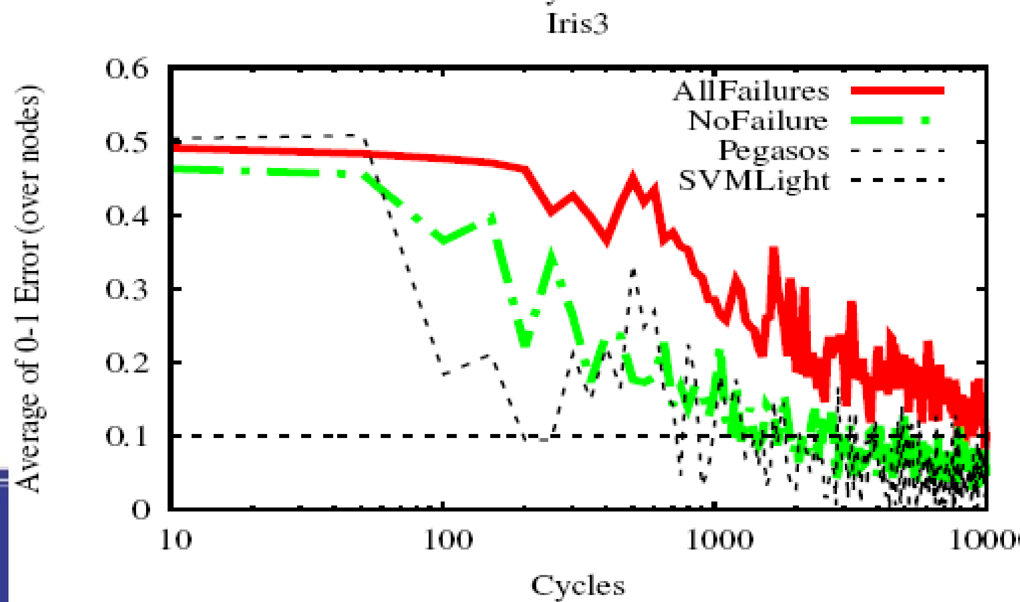
Effects of Different Failures



Size vs. Learnability

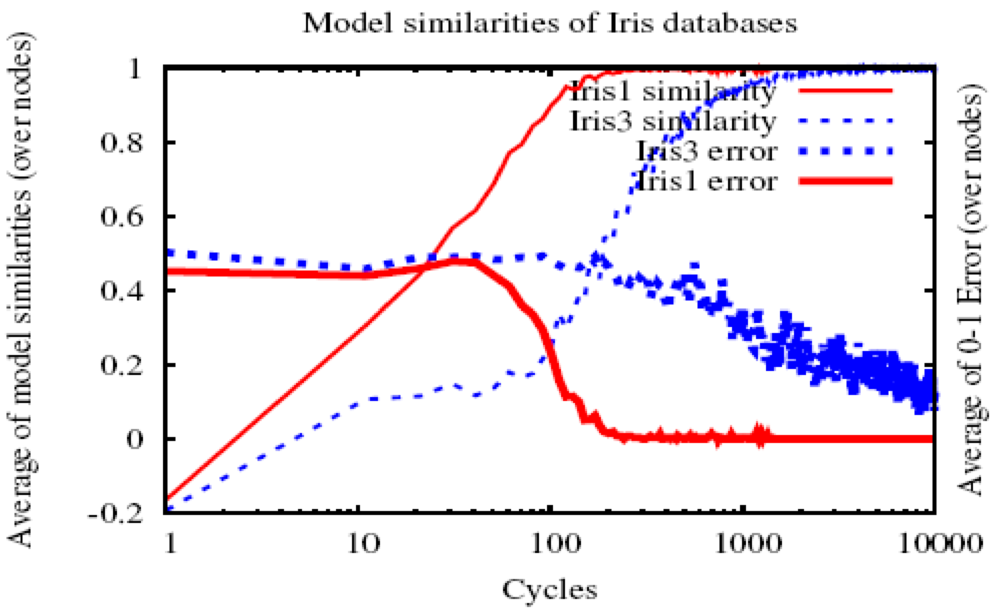
- No relation between DB size and learnability
- Learnability depends on DB patterns rather than the size



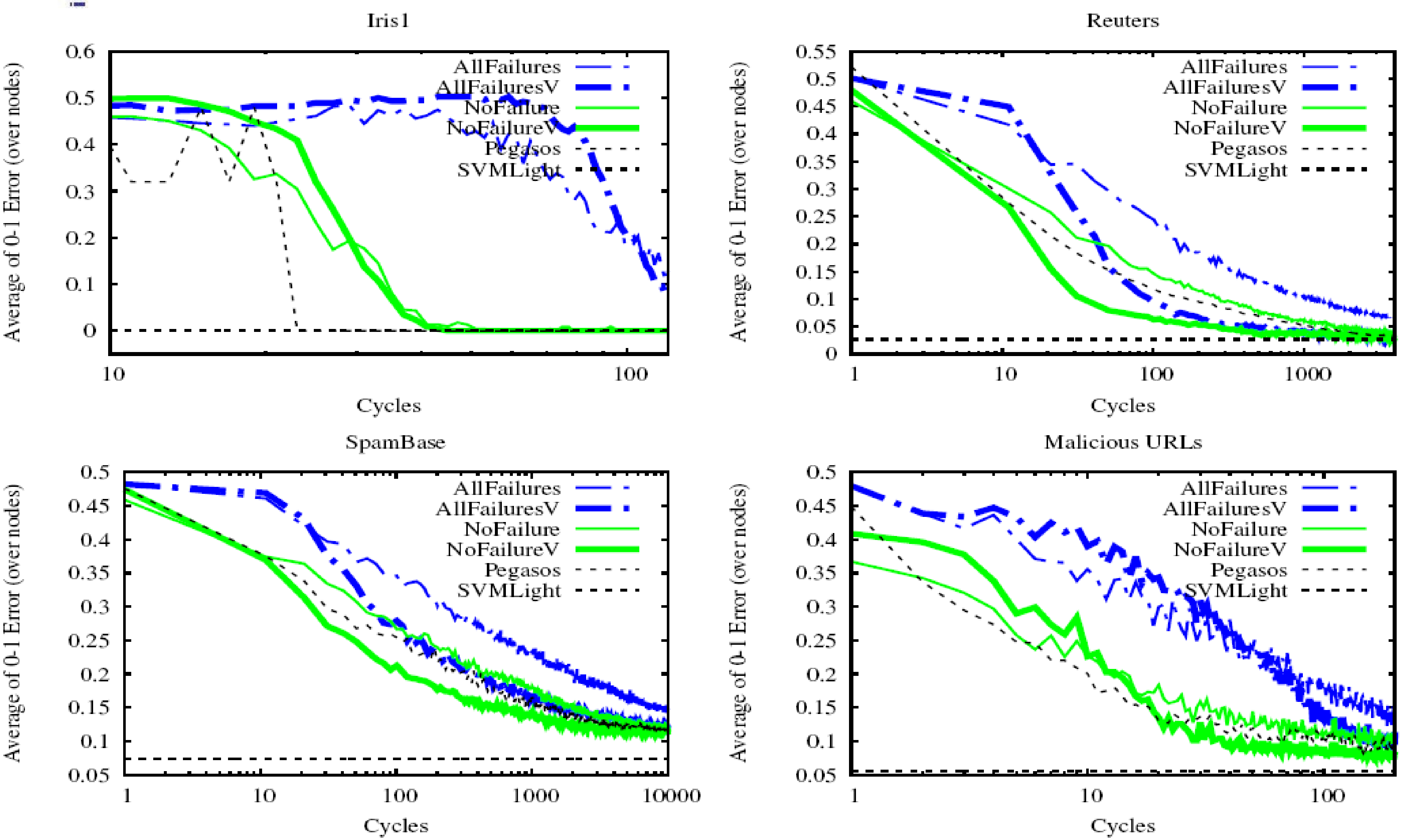


Convergence

- Clearly shows the correlation between model performance and similarity
- The model similarity grows along with the performance
  - the models converge to the same optimum rather than get only more similar



Results on Large Datasets



Summary

- P2P SGD framework was presented
- P2P Pegasos SVM algorithm was implemented and tested on different datasets
- Almost the same performance in a P2P environment as in a centralized one
- The algorithm works well with extreme communication failures as well