*ICT and the Society of the Future (FuturICT.hu)*
*TÁMOP-4.2.2.C-11/1/KONV-2012-0013*

ÚJ SZÉCHENYI TERV

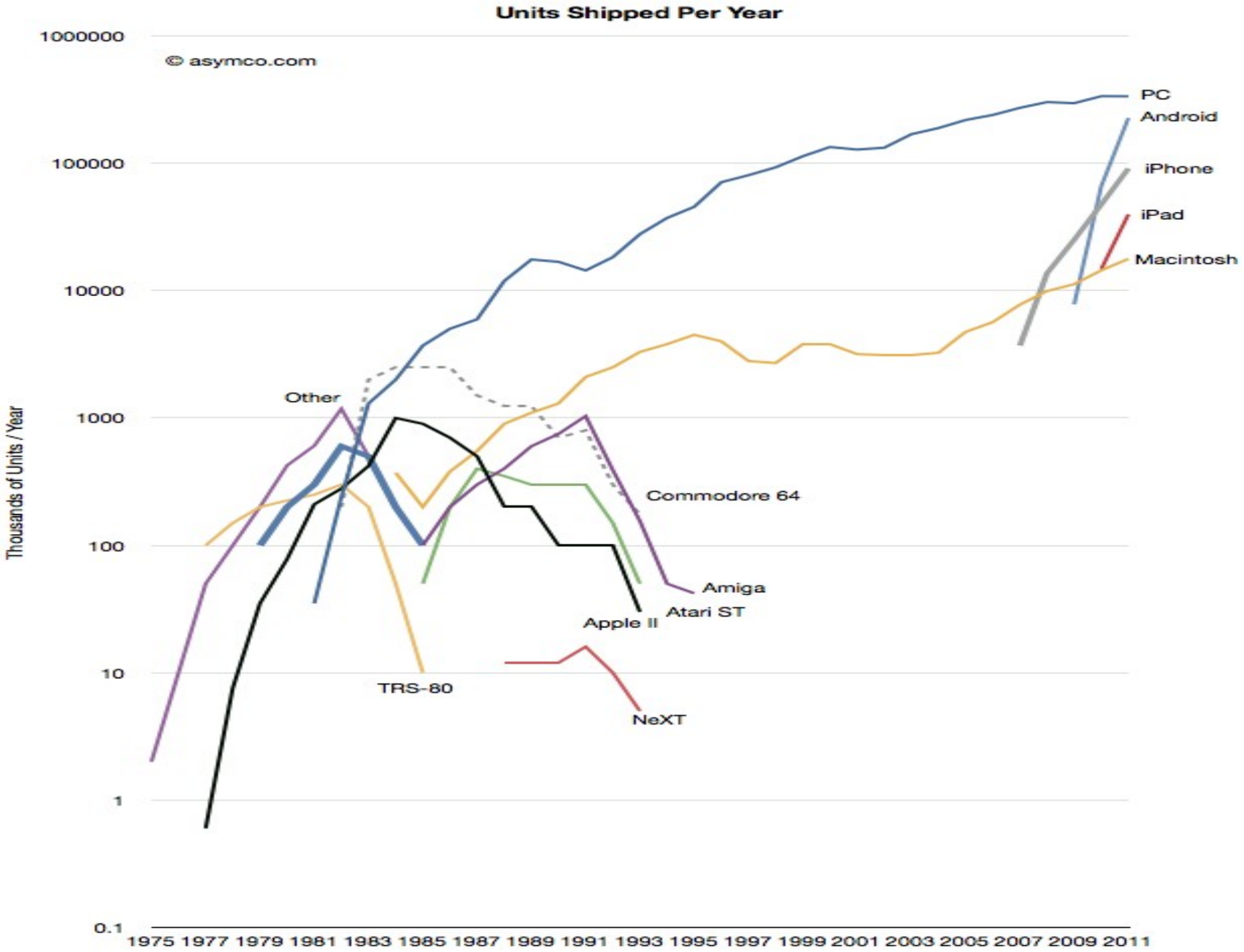
# Towards an open data ecosystem without clouds

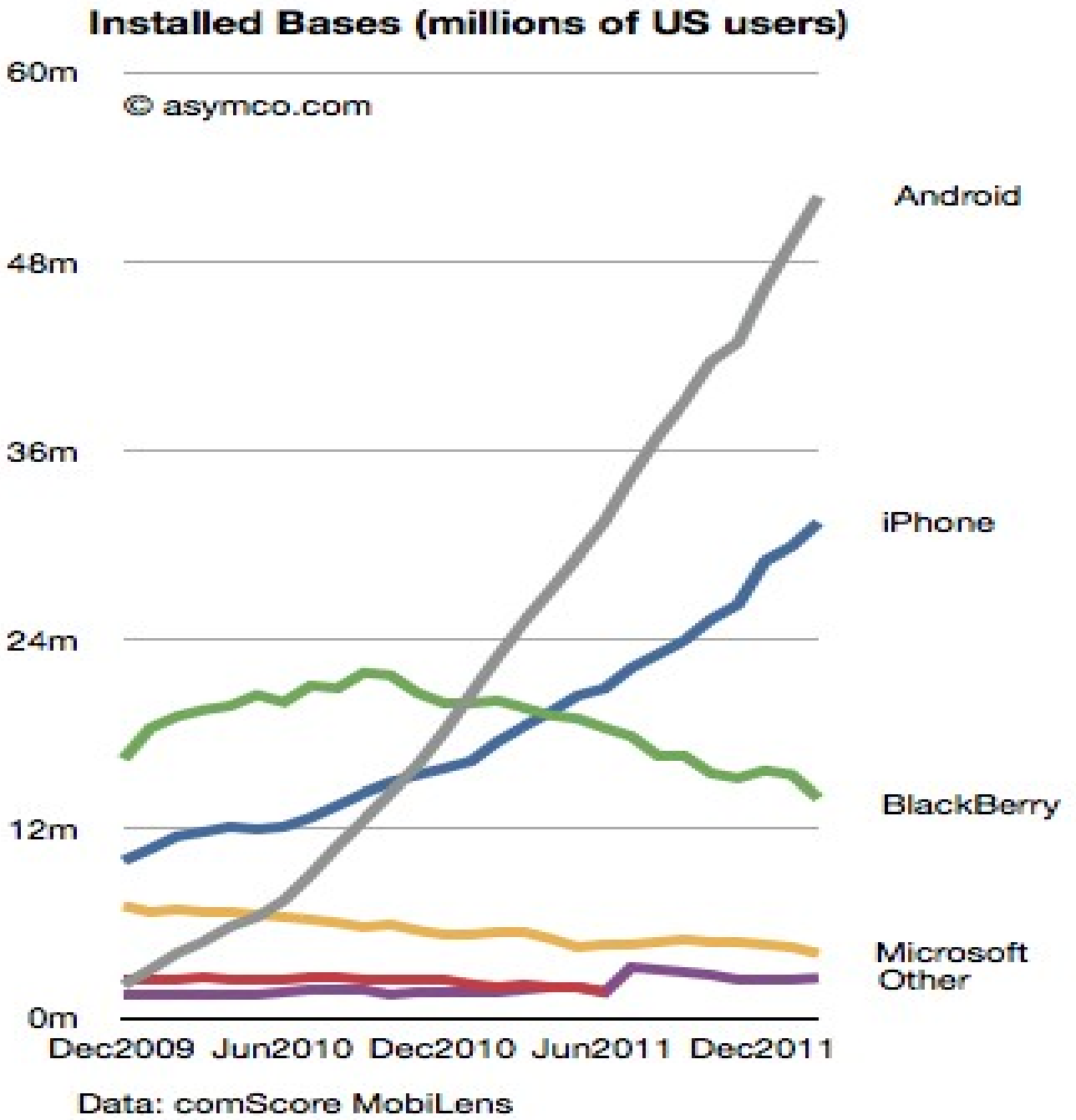
Róbert Ormándi
István Hegedűs
Márk Jelasity

University of Szeged, Hungary

# Outline

- Motivation: the open data ecosystem
- Brief notes on the systems issues
- Massively distributed machine learning: the gossip learning framework (GoLF), illustration through linear SVM
- Low rank matrix factorization in GoLF


Units Shipped Per Year
© asymco.com
Thousands of Units / Year
1000000
100000
10000
1000
100
10
1
0.1
1975 1977 1979 1981 1983 1985 1987 1989 1991 1993 1995 1997 1999 2001 2003 2005 2007 2009 2011
PC
Android
iPhone
iPad
Macintosh
Other
Commodore 64
Amiga
Atari ST
Apple II
TRS-80
NeXT



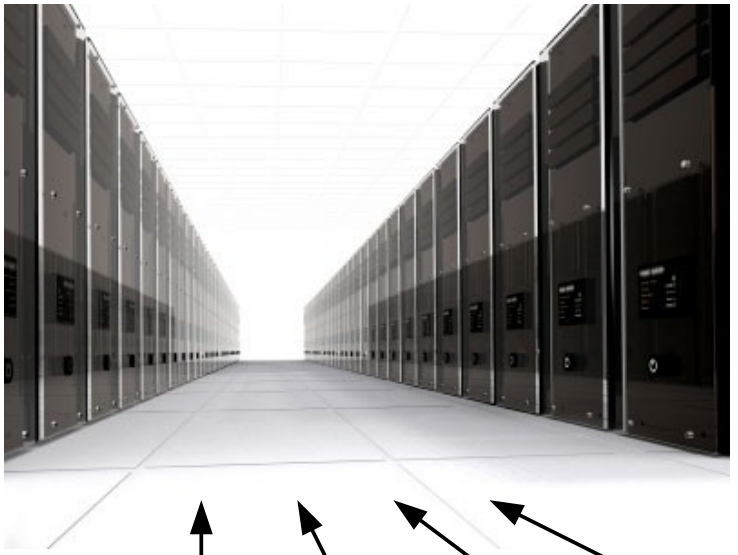

Number of smart phones has passed one billion last October

# Big Data today

Data is owned privately ("closed source")

Storage and processing is costly and problematic (privacy)

So data can serve limited applications (advertising, recommendation, etc)



Private Cloud Infrastructure

# Sci-Fi (as of today)

- Anyone can work with comprehensive global “big” data available in billions of smart phones, and create data models (arbitrary features or predictive models)
- Privacy is fully respected
- Efficient and easy-to-use libraries and schemes are available to support this
- These models are shared and can be used to build further models (like in open source model)
- The cost of entry and cost of maintenance is zero

# So, what if data was open?

- Science (e.g. health)
  - E.g. test the influence of physical activity on chance of getting the flu (*building on models for recording physical activity and probability of flu infection*)
- Business
  - E.g. car accident prediction (*training data based on accident detection, features can include flu infection, current mood (see below), etc*)
- Fun
  - Sky is the limit... E.g. mood predictor, etc

# Outline

- Motivation: the open data ecosystem
- Brief notes on the systems issues
- Massively distributed machine learning: the gossip learning framework (GoLF), illustration through linear SVM
- Low rank matrix factorization in GoLF

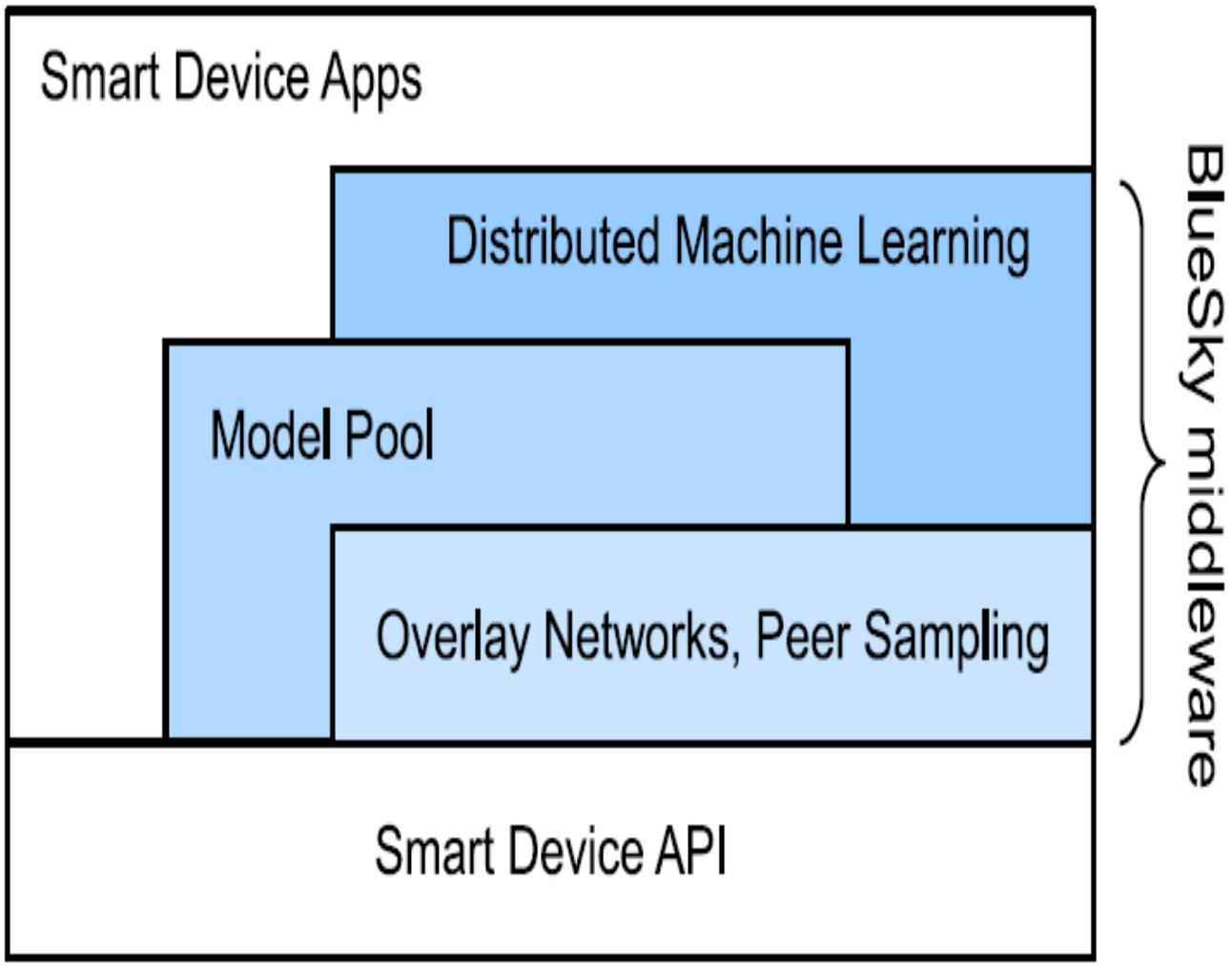
# System model

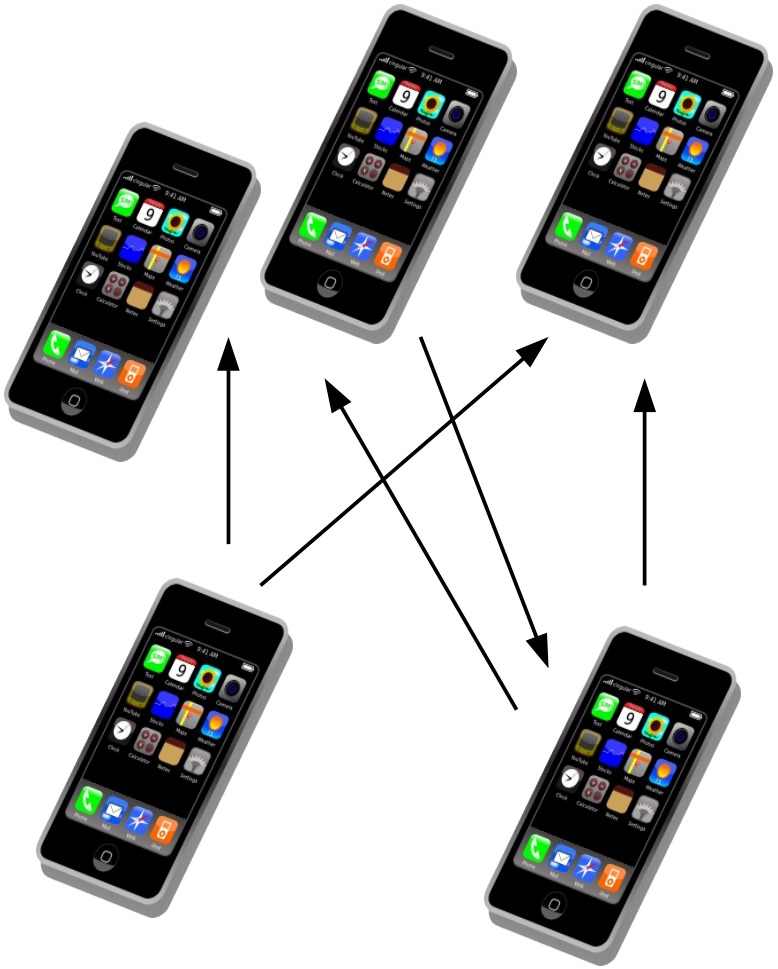
- Large number (millions or more) computers (nodes)
- Packet switched communication
  - Every node has an address
  - Any node can send a message to any given address
    - **That is, almost: in reality, NATs, firewalls**
  - Delay tolerant networks, ad hoc networks, etc are also interesting, but here we will assume any-to-any
- 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
- Data can change (data streams from sensors, or concept drift)
- We do not allow for moving (uploading, collecting, sharing) data, only local processing (helps privacy preservation)

# BlueSky middleware



# Outline

- Motivation: the open data ecosystem
- Brief notes on the systems issues
- Massively distributed machine learning: the gossip learning framework (GoLF), illustration through linear SVM
- Low rank matrix factorization in GoLF

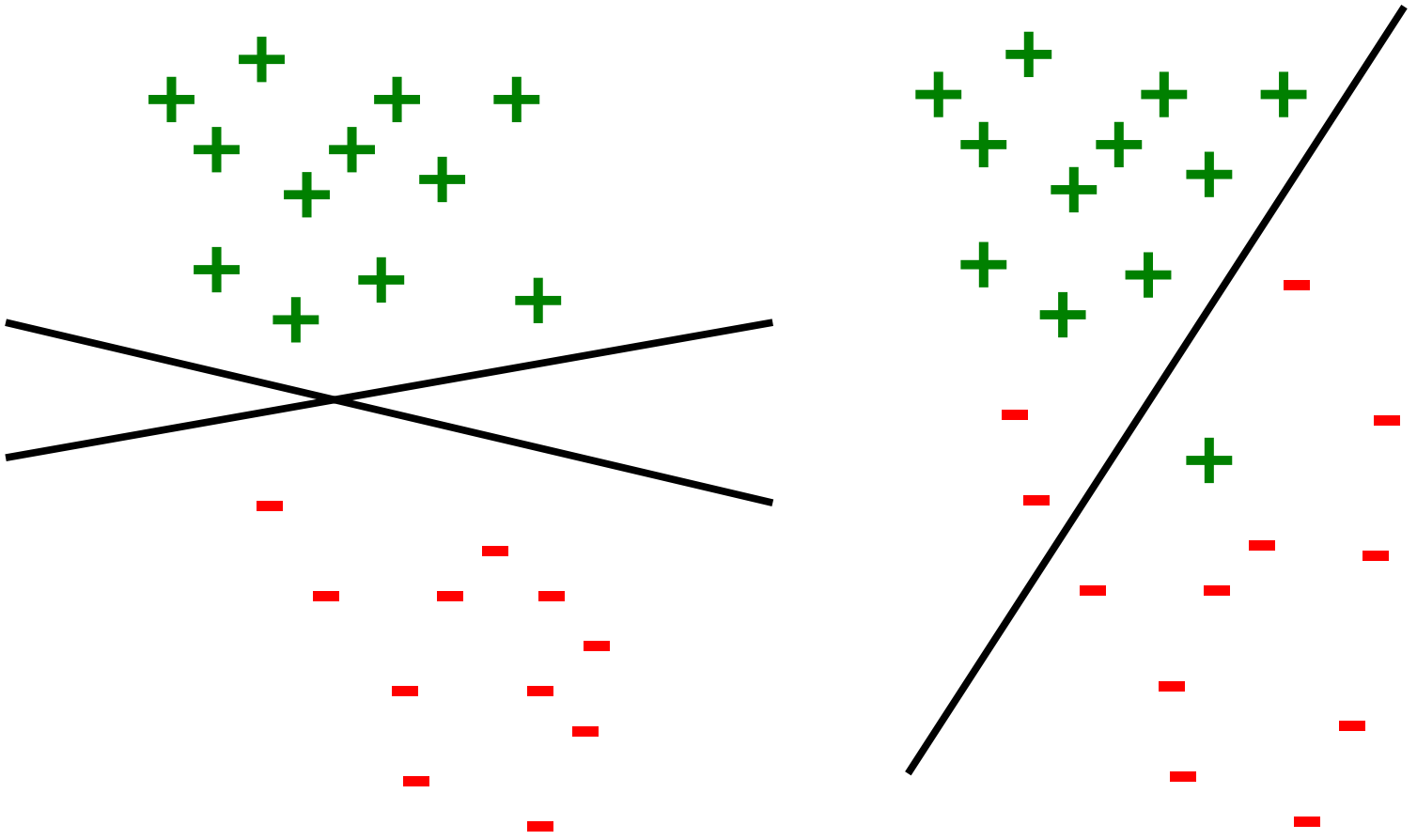
# Massively distributed data mining

- The massively distributed model is limited in computing abilities, but not as much as it seems!
- Asynchronous distributed numeric algorithms exist that
  - Are very fast
  - Are very simple to implement and reason about
  - Provide almost exact, or often exact results in the face of chaotic communication
- E.g. chaotic power method, gossip based aggregation
- 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



# Learning scheme: Walking models, static data

- 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
  - They update the model based on one data record at a time
- Idea 1: models walk over overlays and get updated at local nodes using a suitable online algorithm
- Idea 2: we can combine models that visit the same node

# 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

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

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

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

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

# Gossip learning

**Algorithm 1** Gossip Learning Scheme

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     currentModel ← createModel()
6:     send currentModel to p
7: end loop
8:
9: procedure ONRECEIVEMODEL(m)
10:    modelQueue.add(m)
11: end procedure
```

```
1: procedure CREATEMODELRW
2:     m ← modelQueue.first()
3:     update(m)
4:     return m
5: end procedure
6:
7: procedure CREATEMODELMU
8:     m1 ← modelQueue.first()
9:     m2 ← modelQueue.second()
10:    m ← merge(m1, m2)
11:    update(m)
12:    return m
13: end procedure
```

# The merge function for linear models

- Let z=merge(x,y)=(x+y)/2 (x and y are linear models)
- In the case of linear regression (a.k.a. Adaline perceptron)
  - Updating z using an example has the same effect as updating x and y with the same example and then averaging these two updated models
    - **Update is distributive over merge**
  - Making predictions using z is the same as calculating the weighted average of the predictions of x and y
    - **Prediction is distributive over weighted vote**
  - So, we effectively propagate an exponential number of models, and the voting of these is our prediction
- For other linear models, like the linear SVM, this is only a heuristic argument

# Local prediction

- We use only local models
  - The current model
  - Or voting over a number of recent models

```
1:  procedure PREDICT(x)
2:      w ← currentModel
3:      return sign(⟨w, x⟩)
4:  end procedure

5:  procedure VOTEDPREDICT(x)
6:      pRatio ← 0
7:      for m ∈ modelQueue do
8:          if sign(⟨m.w, x⟩) ≥ 0 then
9:              pRatio ← pRatio +1
10:         end if
11:     end for
12:     return sign(pRatio/modelQueue.size()−0.5)
13: end procedure
```

# Linear SVM

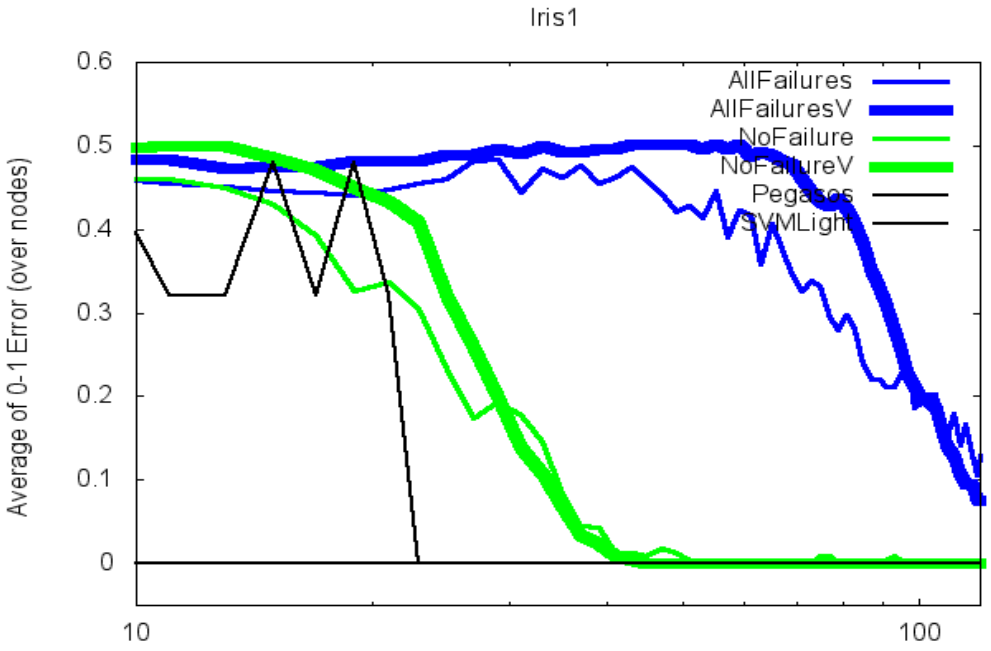
- We plugged in a linear SVM with stochastic gradient (Pegasos by Shalev-Shwartz et al (2010))
- We theoretically proved convergence under merging
- We used several benchmark data sets for evaluations
  - Data is fully distributed: one data point per node
- We used extreme scenarios
  - 50% message drop rate
  - 1-10 cycles of message delay
  - Churn modeled after the FileList.org trace from Delft

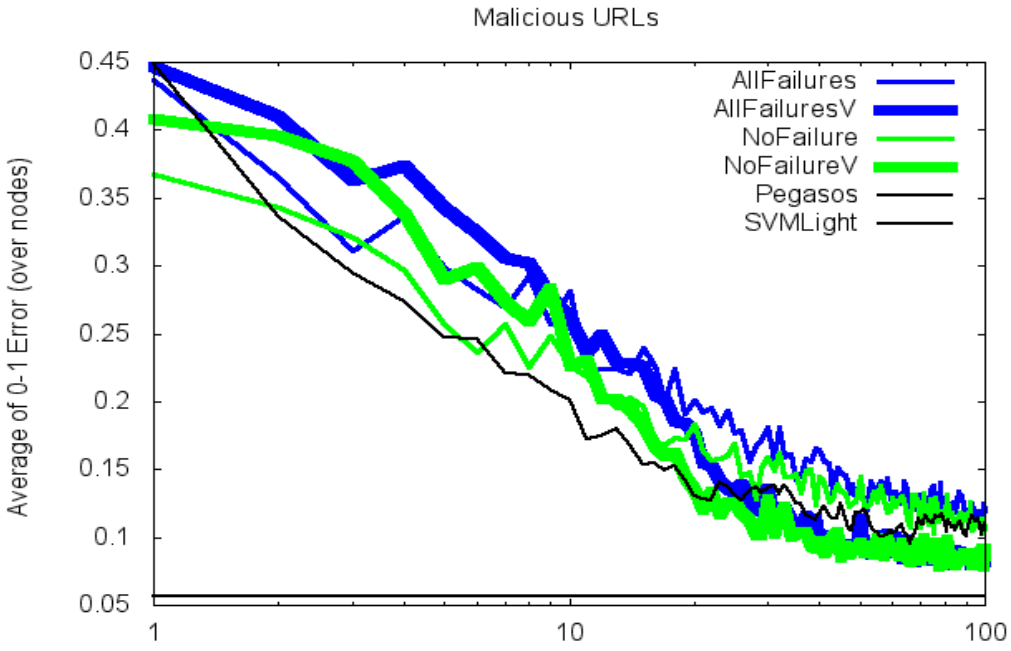
# Data sets

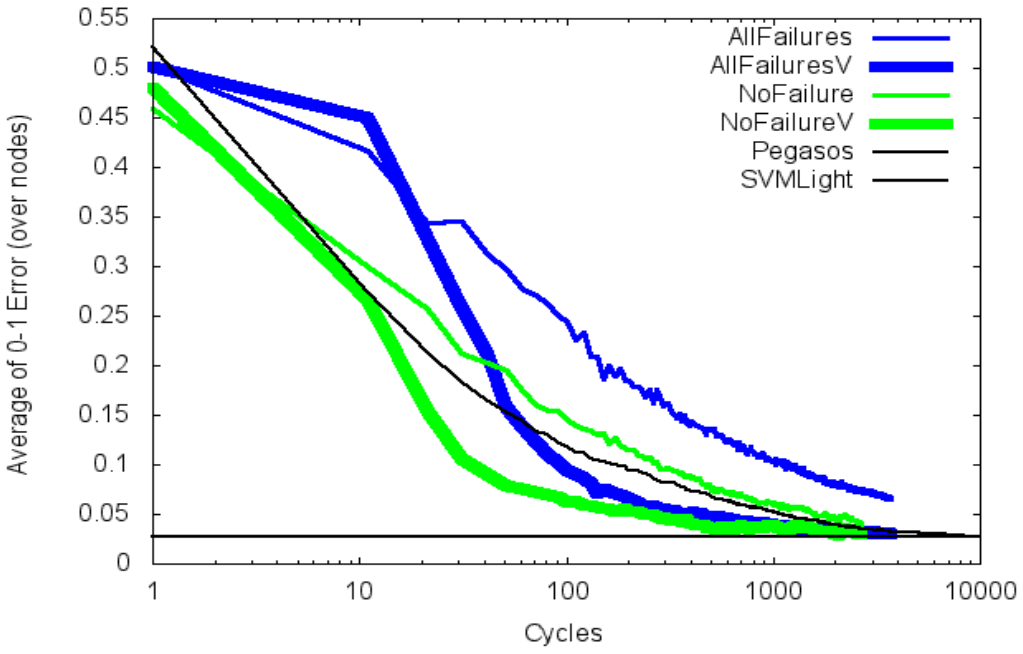
| | Iris1 | Iris2 | Irirs3 | Reuters | SpamBase | Malicious10 |
|---|---|---|---|---|---|---|
| Training set size | 90 | 90 | 90 | 2000 | 4140 | 2155622 |
| Test set size | 10 | 10 | 10 | 600 | 461 | 240508 |
| Number of features | 4 | 4 | 4 | 9947 | 57 | 10 |
| Classlabel ratio | 50/50 | 50/50 | 50/50 | 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 (–) |

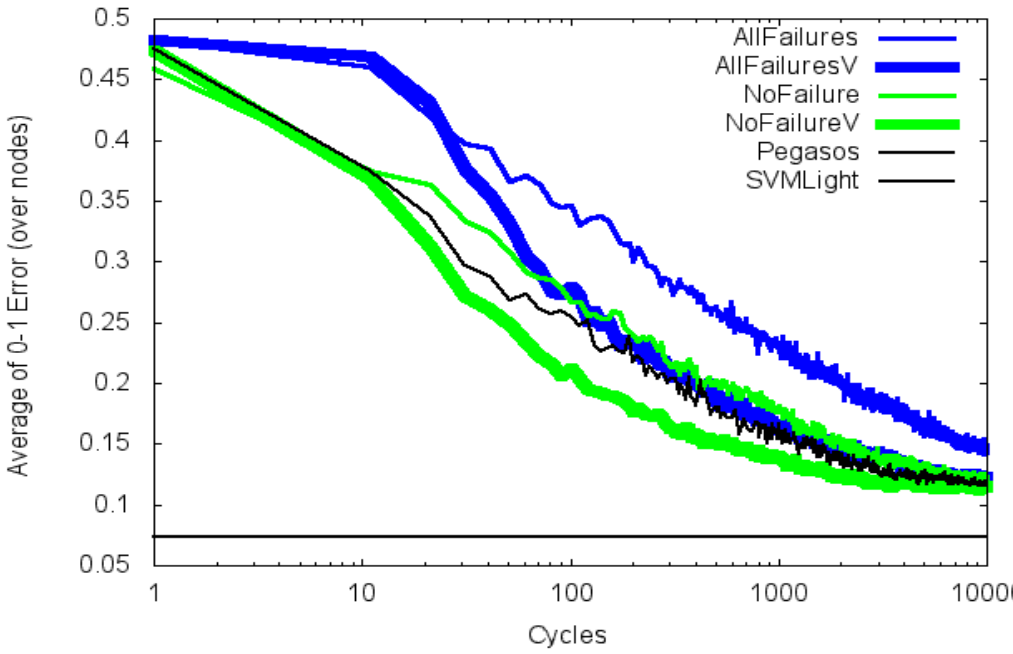
- Statistics of data sets
- The performance of some known algorithms

# Without merge

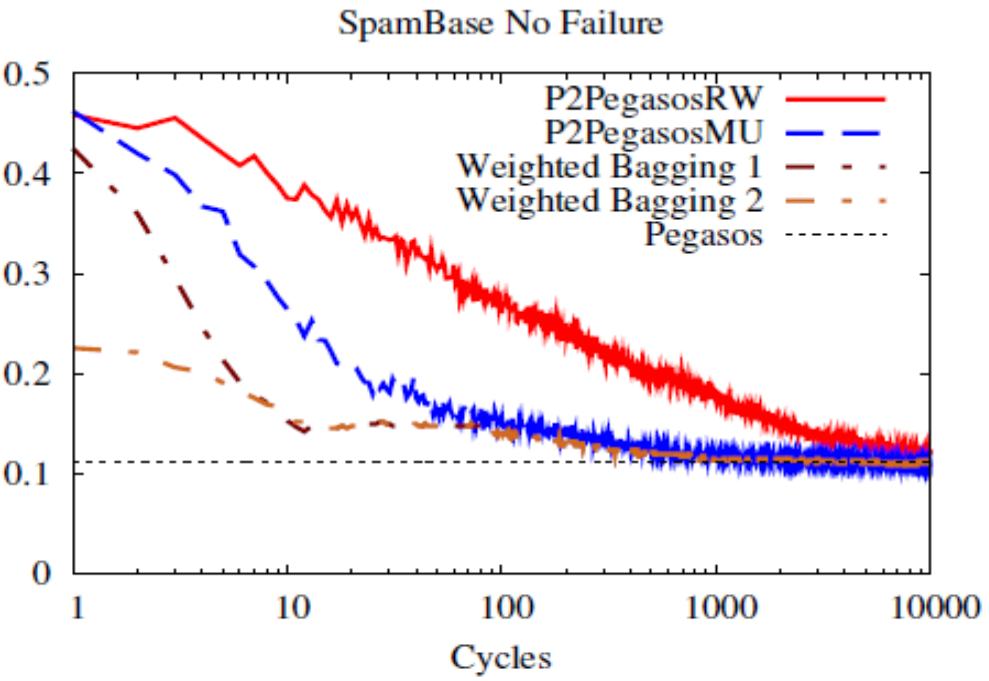


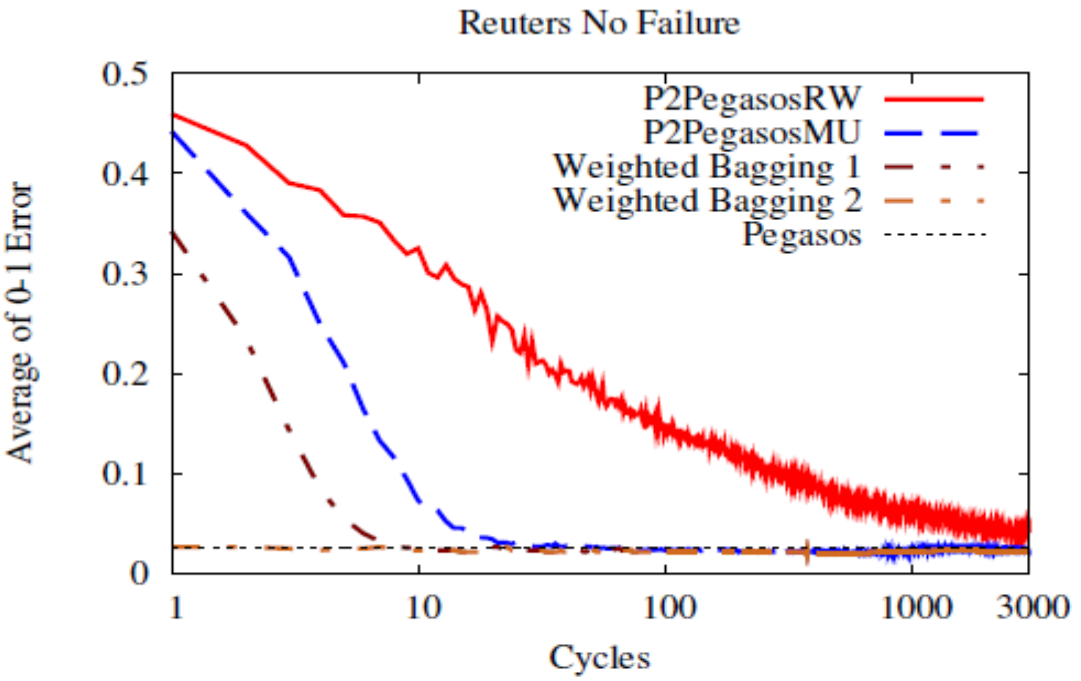



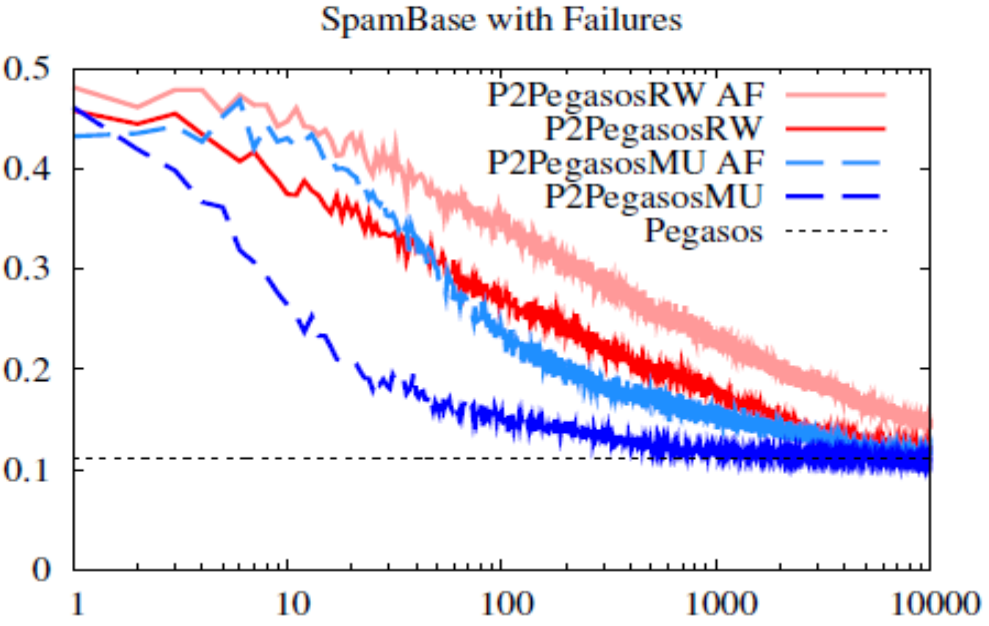



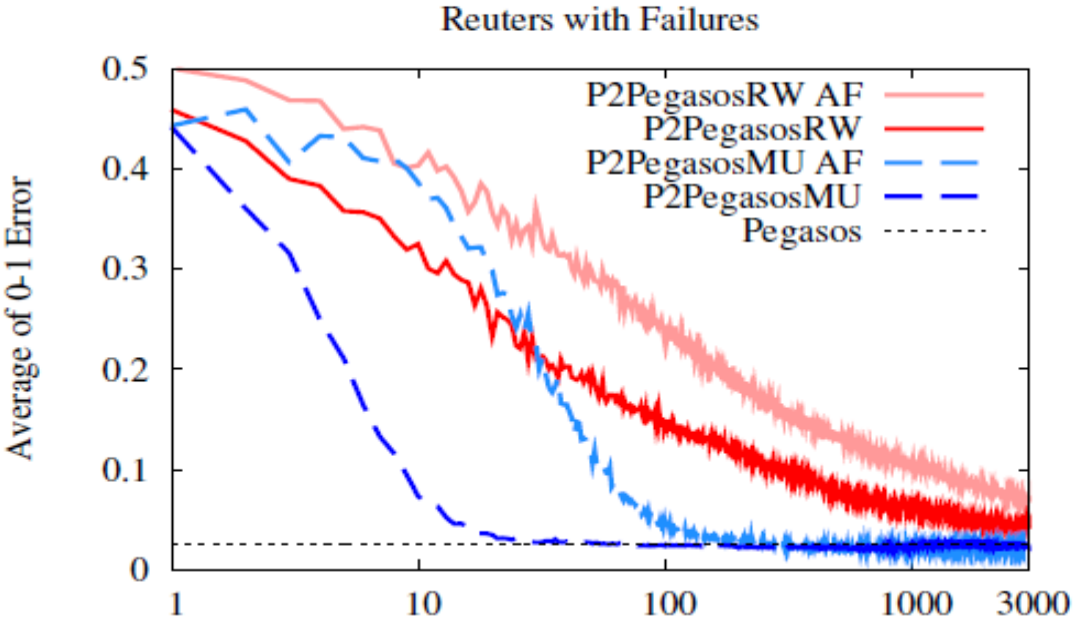



# With merge


SpamBase No Failure
P2PegasosRW
P2PegasosMU
Weighted Bagging 1
Weighted Bagging 2
Pegasos
0.5
0.4
0.3
0.2
0.1
0
1
10
100
1000
10000
Cycles


Reuters No Failure
P2PegasosRW
P2PegasosMU
Weighted Bagging 1
Weighted Bagging 2
Pegasos
Average of 0-1 Error
0.5
0.4
0.3
0.2
0.1
0
1
10
100
1000
3000
Cycles


SpamBase with Failures
P2PegasosRW AF
P2PegasosRW
P2PegasosMU AF
P2PegasosMU
Pegasos
0.5
0.4
0.3
0.2
0.1
0
1
10
100
1000
10000

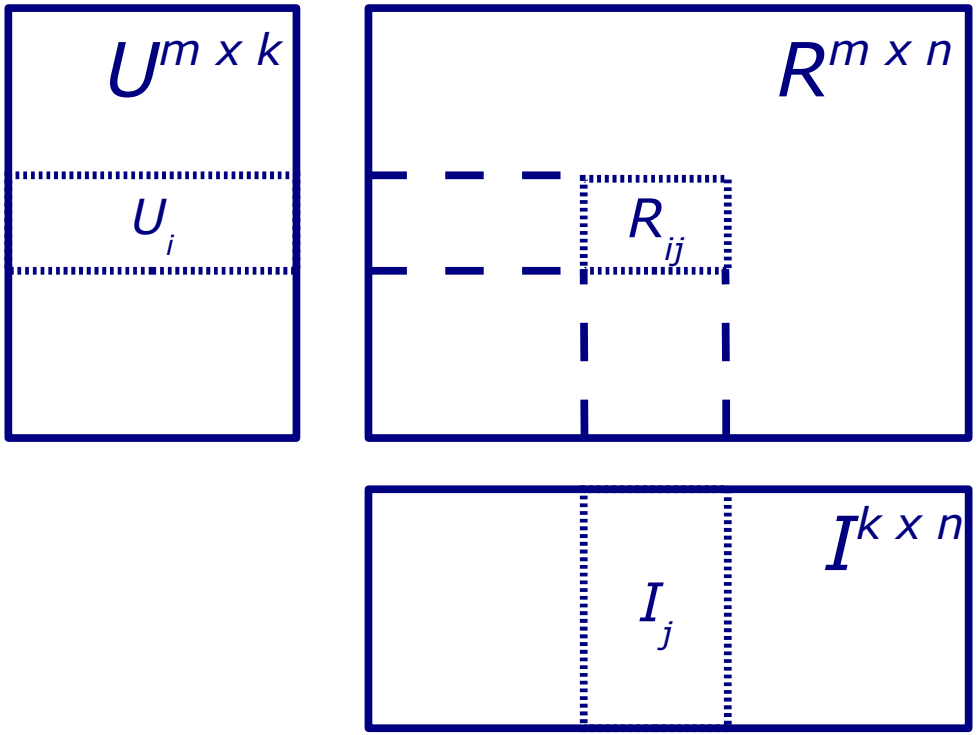

Reuters with Failures
P2PegasosRW AF
P2PegasosRW
P2PegasosMU AF
P2PegasosMU
Pegasos
Average of 0-1 Error
0.5
0.4
0.3
0.2
0.1
0
1
10
100
1000
3000

# Outline

- Motivation: the open data ecosystem
- Brief notes on the systems issues
- Massively distributed machine learning: the gossip learning framework (GoLF), illustration through linear SVM
- Low rank matrix factorization in GoLF

# Low rank matrix approximation

- Work in progress!
- Given matrix R, we want to find U and I such that UI is close to R
- Dimension k is very small: we want compression (eg topic models)
- R is not fully defined (eg non-rated items if R is user-item matrix for recommendation)
- We assume one node has one row of R: private information, eg item ratings




Credits: Levente Kocsis (main idea) and András Benczúr, Inst. for CS and Control, Budapest

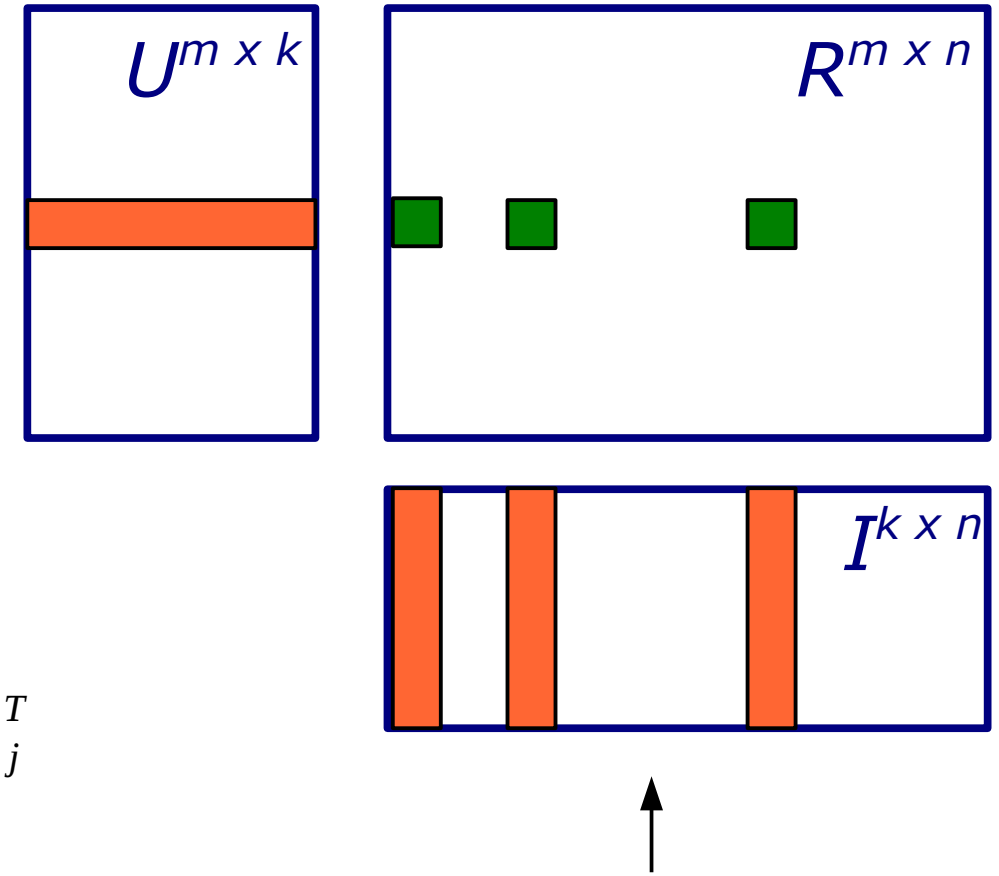
# Low rank matrix approximation

- The model that walks is mx. I
- Rows of U (and R) are local to nodes, they do not move
- $Q_i$ contains rated item indices.
  Ignoring regularization:

$$Err(U_i, I) = \frac{1}{|Q_i|} \sum_{j \in Q_i} (R_{i,j} - U_i I_j)^2$$

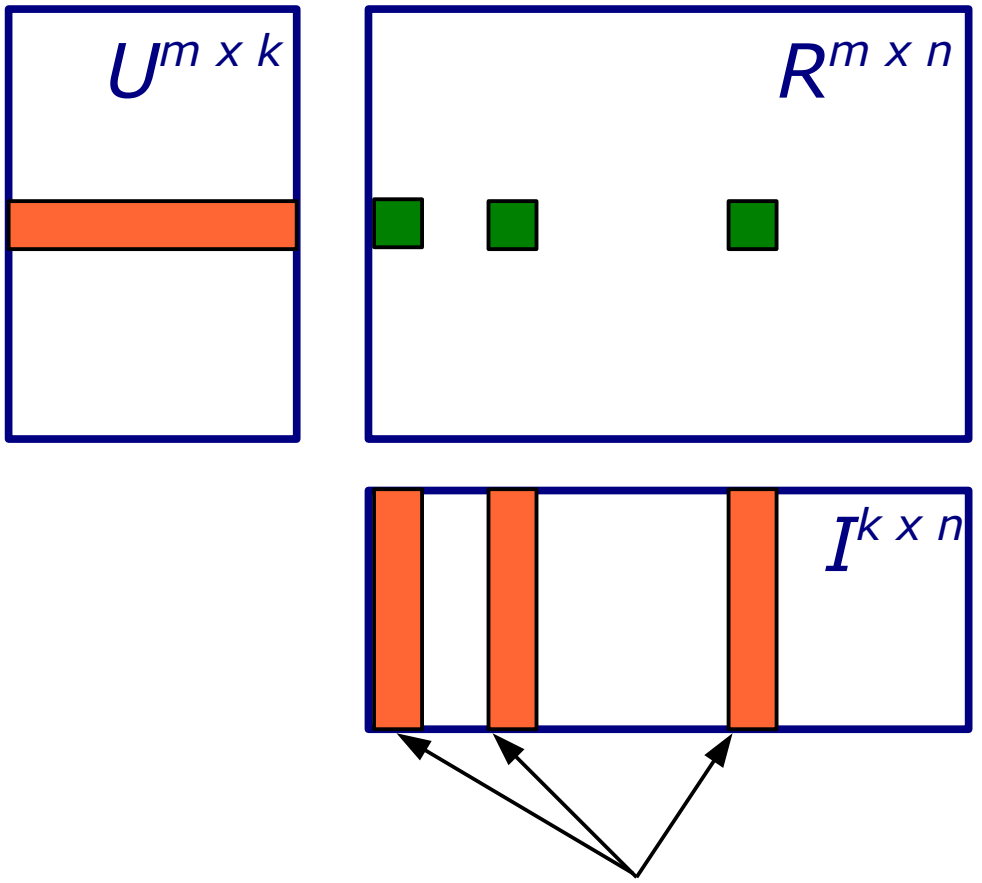
$$\frac{\partial Err(U_i, I)}{\partial U_i} = \frac{2}{|Q_i|} \sum_{j \in Q_i} (R_{i,j} - U_i I_j) I_j^T$$

$$\forall j \in Q_i : \frac{\partial Err(U_i, I)}{\partial I_j} = \frac{2}{|Q_i|} (R_{i,j} - U_i I_j) U_i^T$$



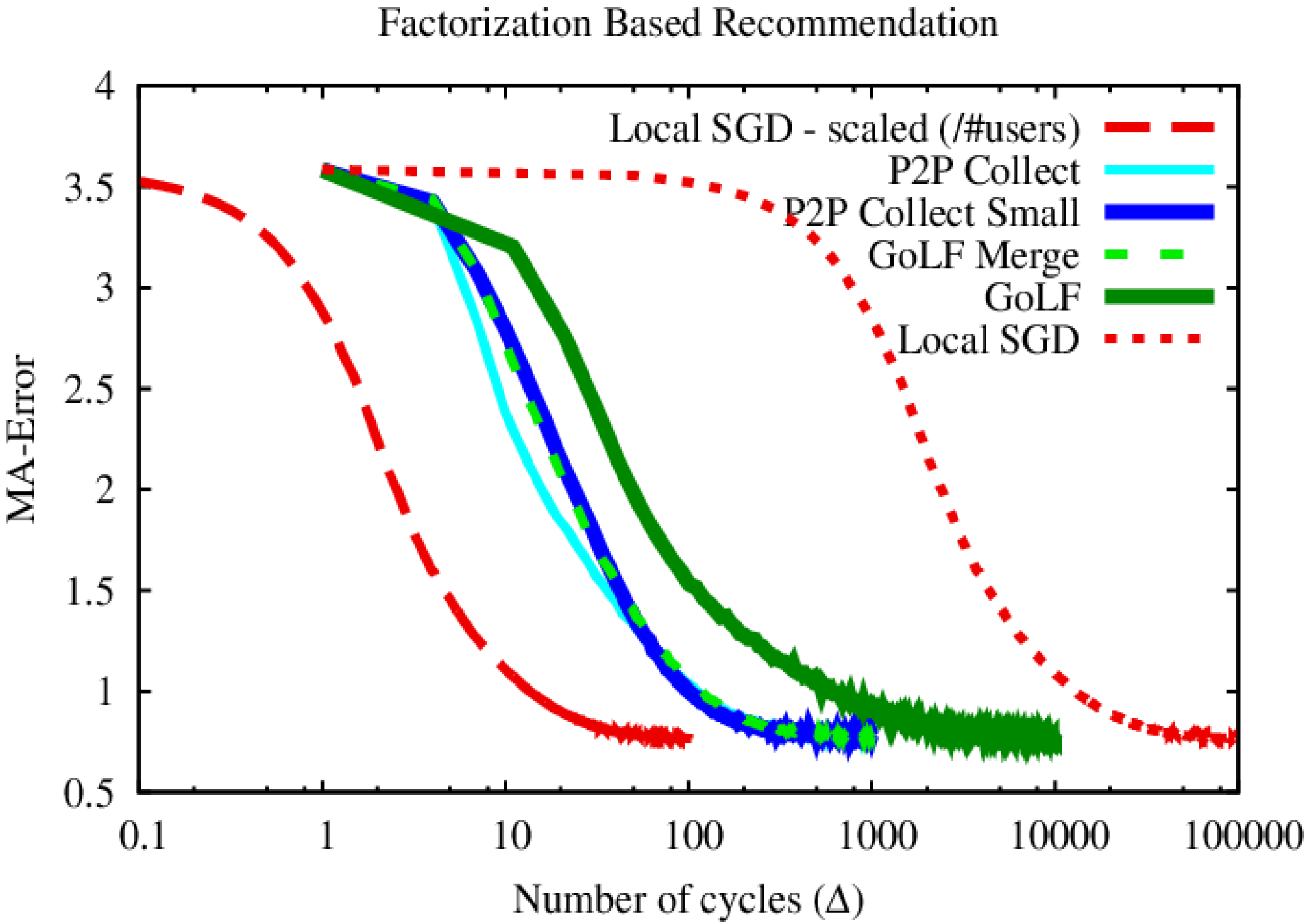
# Low rank matrix approximation

- Matrix I is relatively small, but we might do better
  - propagate only those columns that change
  - Merge received columns to local version of I
- So, here we have a local complete version of I, in which we merge incoming columns as they arrive




Gossip updated columns only

# Small MovieLens (1000 users, 1700 movies, 100,000 ratings)



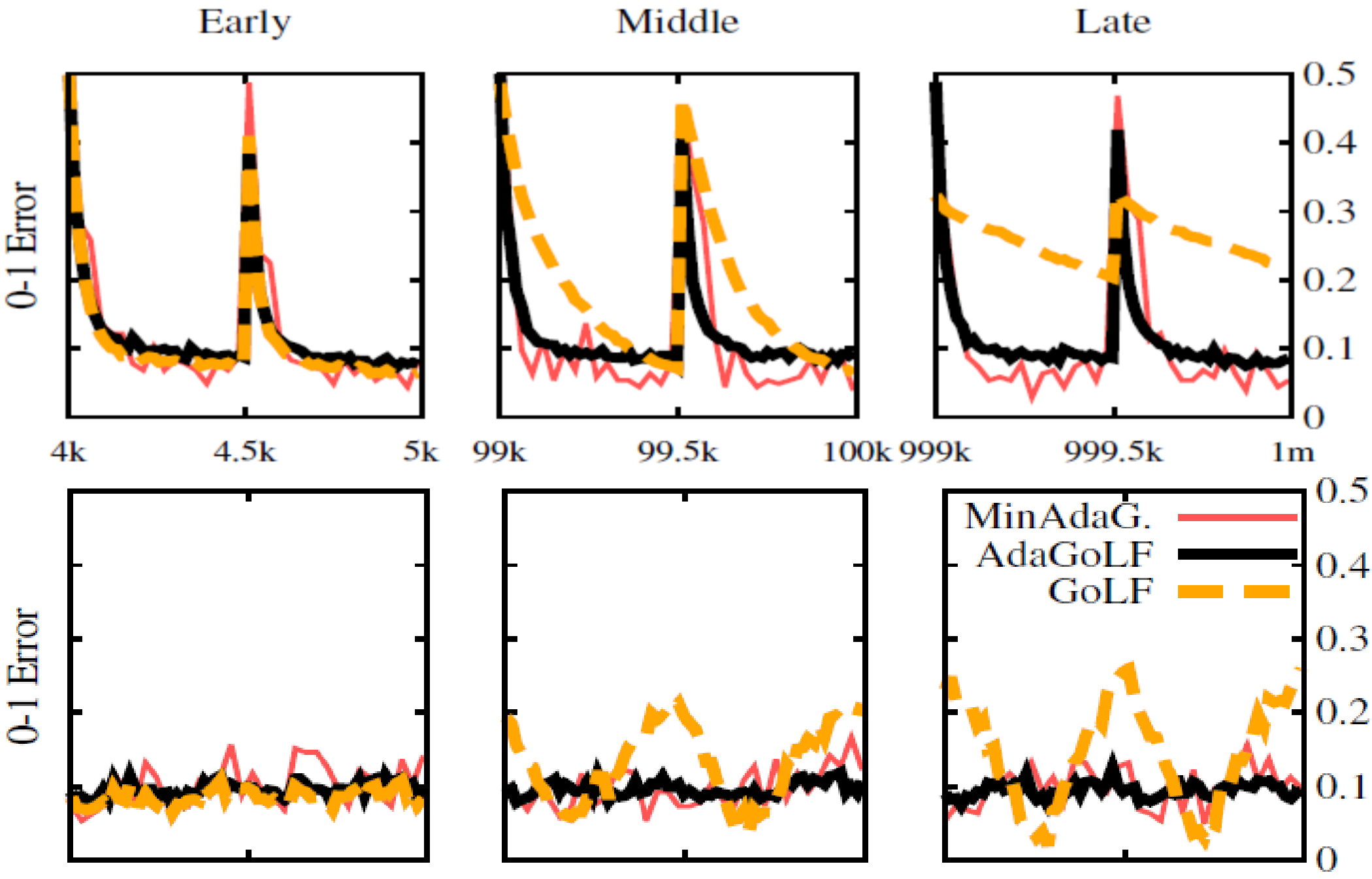
# Publications


- 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,25(4):556–571, 2013. (doi:10.1002/cpe.2858)
- István Hegedűs, Róbert Busa-Fekete, Róbert Ormándi, Márk Jelasity, and Balázs Kégl. Peer-to-peer multi-class boosting. In Euro-Par 2012, volume 7484 of Lecture Notes in Computer Science, pages 389-400. Springer-Verlag, 2012.
- István Hegedűs, Róbert Ormándi, and Márk Jelasity. Gossip-based learning under drifting concepts in fully distributed networks. In Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012), pages 79–88. IEEE Computer Society, 2012.

# Additional results

- We implemented multiclass boosting in the gossip framework
- We developed techniques for dealing with concept drift
  - The algorithm is running continuously
  - We keep the age distribution of models fixed
  - At any point in time we have good models

# The Drifting problem

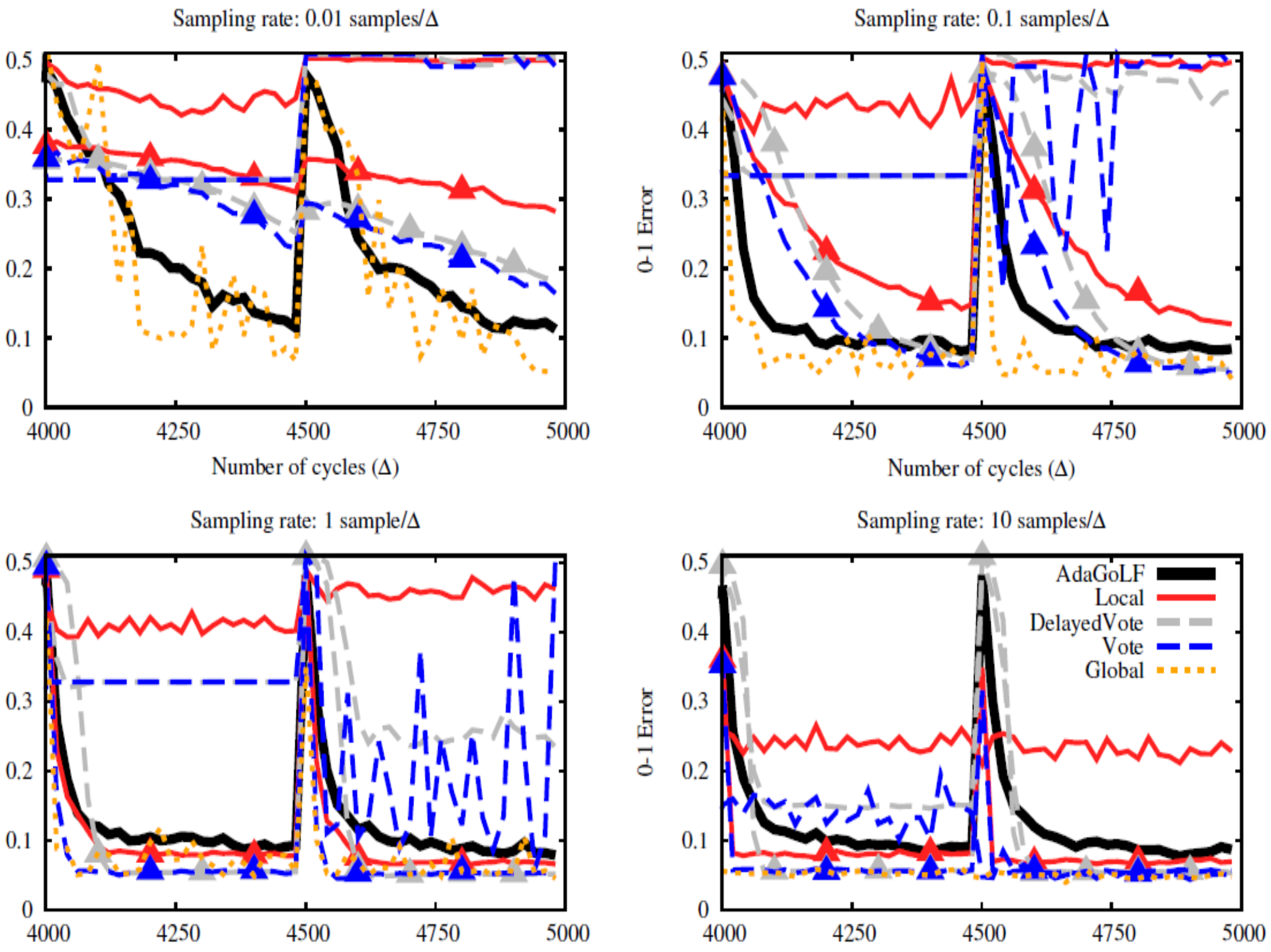


# How to set the TTL

- 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

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

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


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

# Concept Drift 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”