UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
Research Group on Artificial Intelligence

Peer-to-peer (P2P) gépi tanulás

Hegedűs István

Motiváció

- Adatokban rejlő információk kinyerésének fontossága → adatbányászat, gépi-tanulás, modell építés
  - Különböző módszerekkel összegyűjtött adatok feldolgozása
  - Adathalmazokban rejlő minták azonosítása
  - Minták (osztályok) egymástól való elkülönítése
- Lehetséges felhasználási területek:
  - Spam szűrés
  - Vélemény detekció
  - Ajánló rendszerek

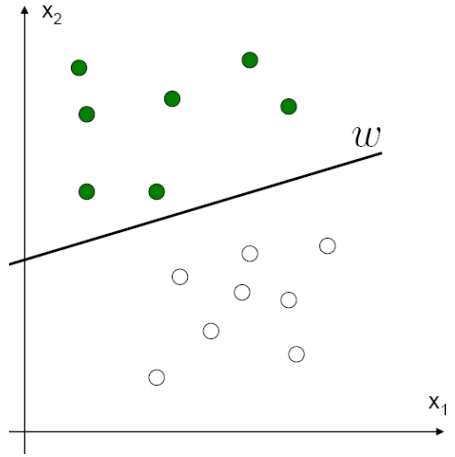
Motiváció

- Napjainkban rengeteg információ keletkezik „lokálisan” (szenzor hálózatok, „okos” telefonok)
  - Adatok „centralizálása” → modell építése
  - Modell építése centralizálás nélkül → p2p pletyka alapú algoritmusok alkalmazása
    - (személyes információk a lokális gépen maradnak ⇔meglévő algoritmusainkat át kell alakítanunk)

Rendszer- és adatmodell

- Adott számítógépek (node, peer) egy hálózata
- Az adatbázis el van osztva a hálózaton
  - Node-onként egy vagy csak néhány példa
- A node-ok el tudják kérni egy véletlenszerűen választott node címét a hálózatban
  - a NewsCast nevű protokoll segítségével
- Egy node üzenetet tud küldeni egy másik node-nak, ha ismeri a címét
- Cél: modell építése üzenete segítségével, de lokális számítást alkalmazva

Osztályozás



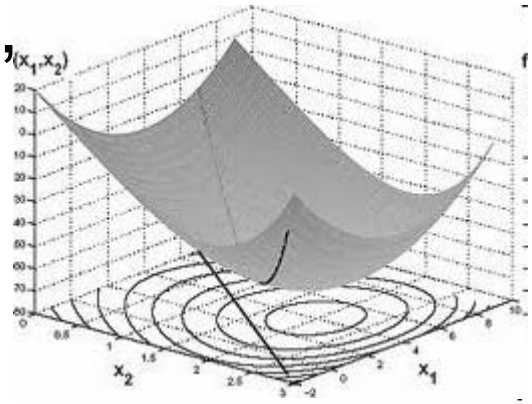
- Két-osztályos eset
  - Adottak $(x_1, y_1), \ldots, (x_n, y_n)$ tanító példák, ahol $x_i \in \mathbb{R}^d$ és $y_i \in \{-1, 1\}$
  - Feladat: keresünk egy modellt $f : \mathbb{R}^d \to \{-1, 1\}$,amely helyesen szét tudja választani a különböző osztályba tartozó példákat → minimalizálja az alábbi formulát

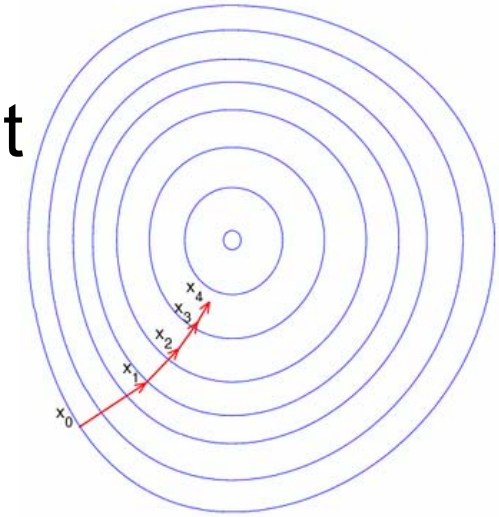
$$\min_f \sum_i (f(x_i) - y_i)^2 \quad i = 1, \ldots, n$$

  - Lineáris esetben a modell egy d dimenziós hiperpsíknak (w) felel meg
- Egy lehetséges módszer a keresett modell megtalálására a sztochasztikus gradiens módszer (SGD)

Stochastic Gradient Descent (SGD) centralizált eset

- A SGD egy optimalizálási módszer, amely egy előre definiált függvény ($(f(x_i) - y_i)^2$) minimumának a helyét próbálja megtalálni gradiens lépések sorozatával.
- Minden egyes iterációban egyetlen véletlenül választott példa segítségével kiszámolja az gradiens lépést és frissíti a modellt
- Miért SGD: mivel iterációnként elegendő egy példa az egész adatbázis helyett





Stochastic Gradient Descent (SGD) centralizált eset

- Legyen a hibafüggvény

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

- Ekkor a gradiens

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

- Így a gradiens alapú frissítés

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

- De mivel az SGD egy példa alapján frissít

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

P2P SGD

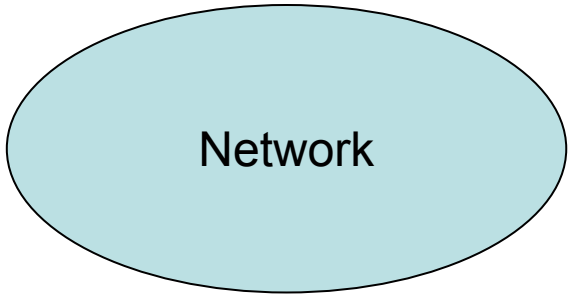
- Ötlet: a modellek node-ról node-ra „ugrálnak” és frissítik magukat az ott található tanító példa segítségével
- Az algoritmus megegyezik az eredeti SGD algoritmus egy „kifordított” változatával
  - Ha tudjuk garantálni a mintázás véletlenszerűségét, akkor az algoritmusunk ugyan abba az optimumba konvergál
- További előnyök: az adatok sosem hagyják el a node-okat, személyes, érzékeny információk helyben maradnak

P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

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

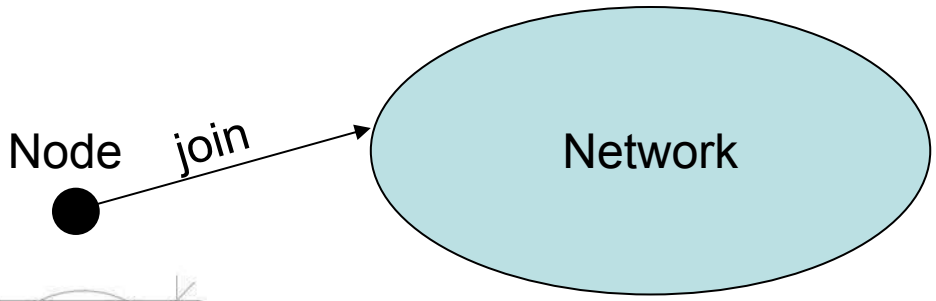
Node



P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

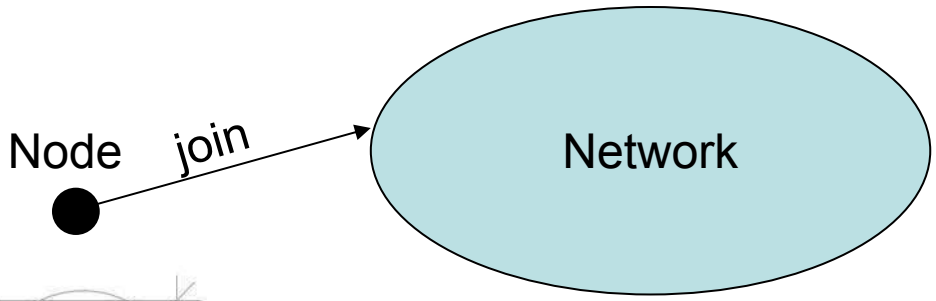
```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: procedure onReceiveModel(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```



P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

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

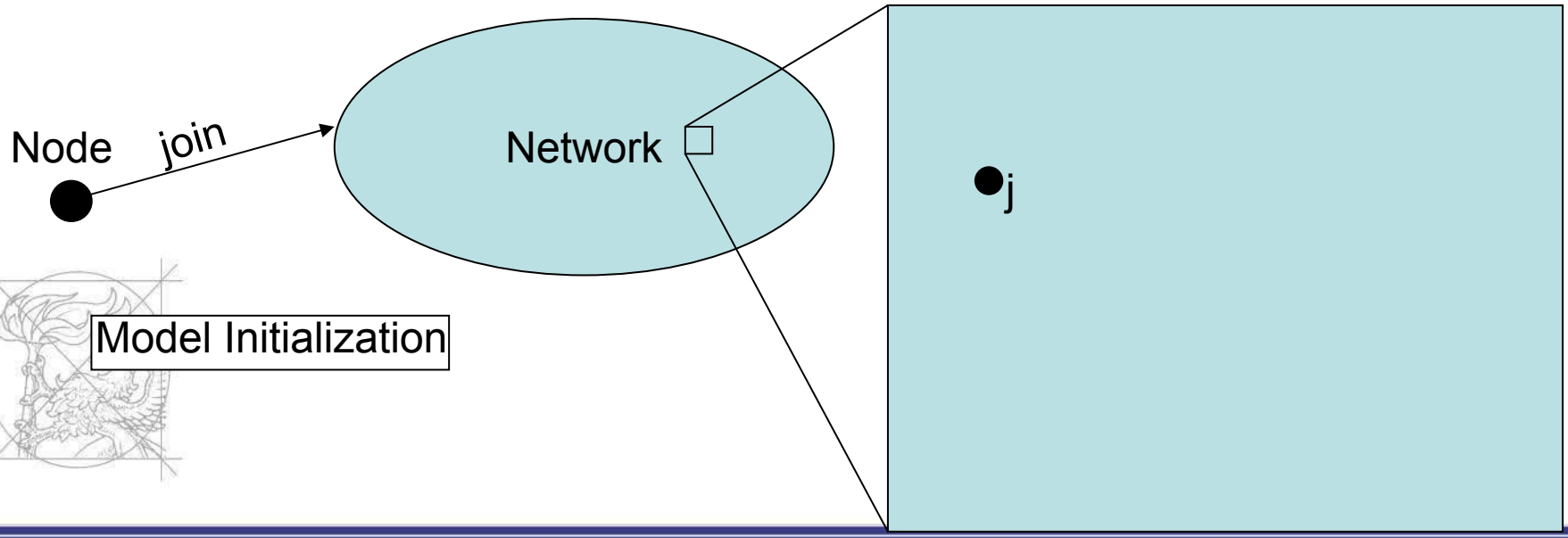


Model Initialization

P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

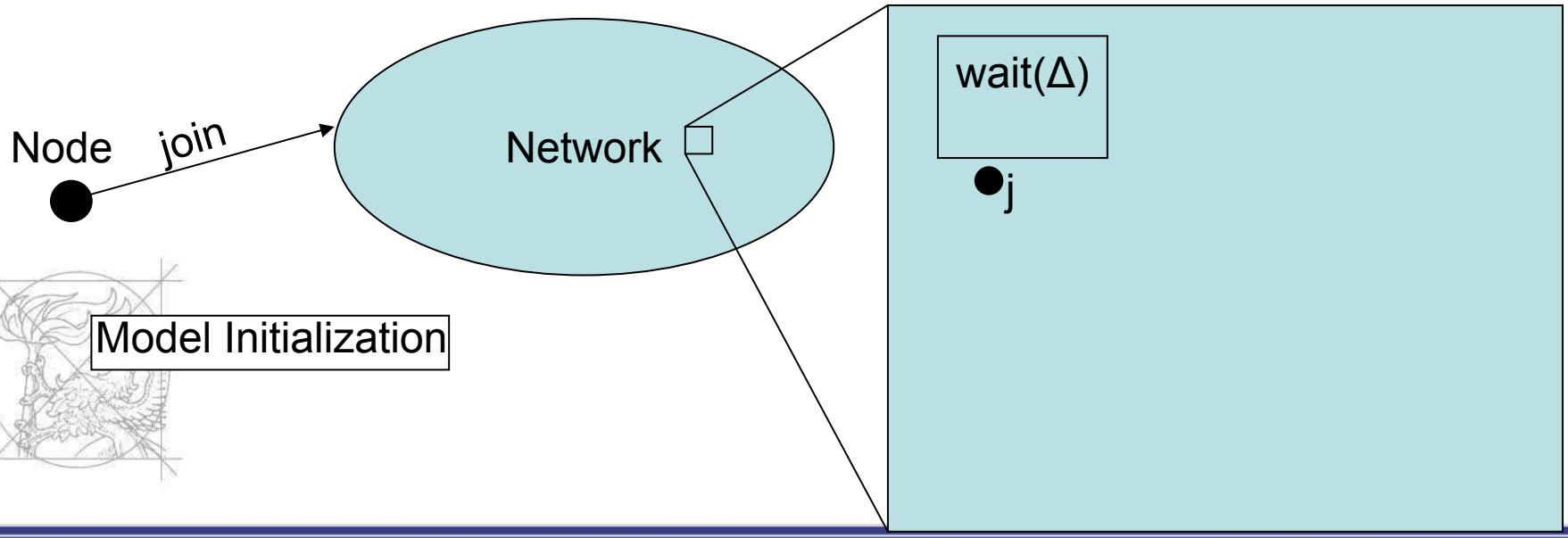
```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: procedure onReceiveModel(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```



P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

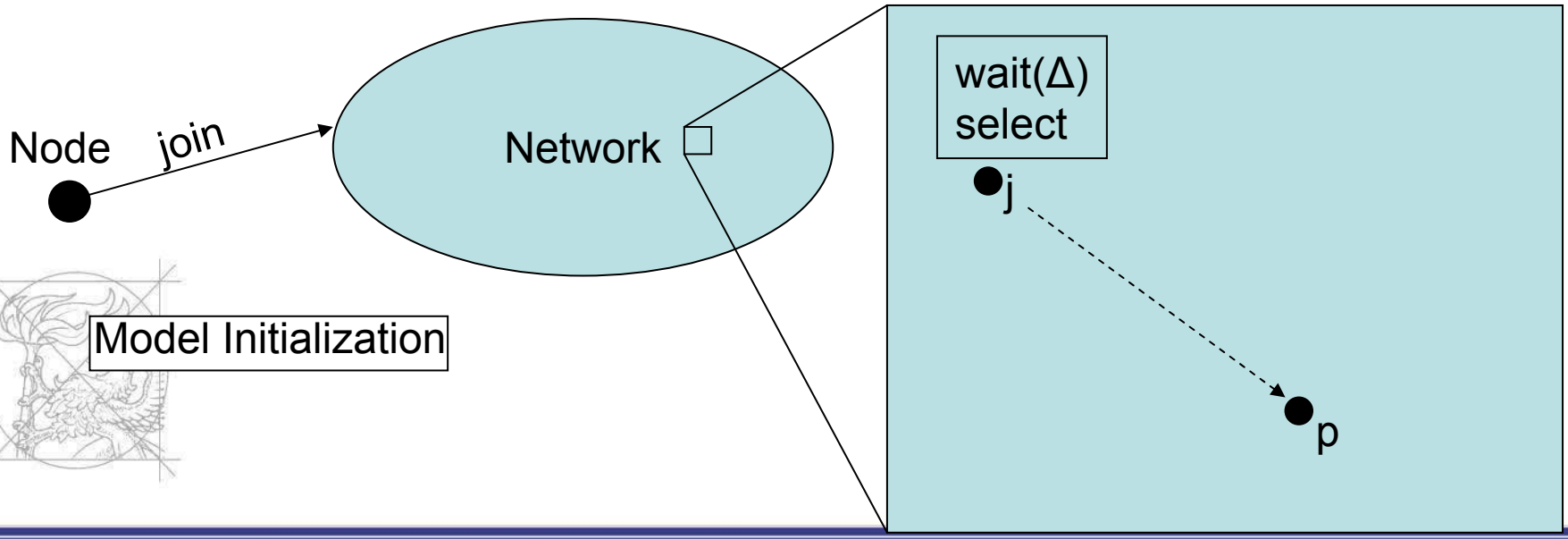
```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: procedure onReceiveModel(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```



P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

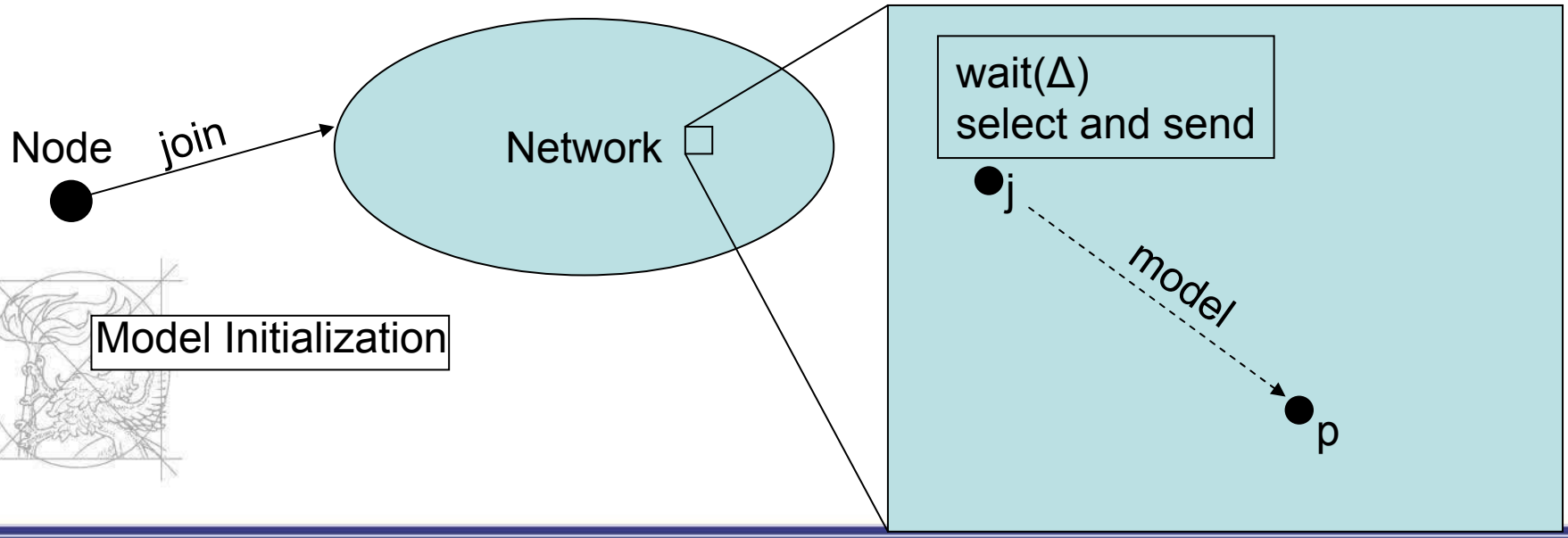
```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: procedure onReceiveModel(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```



P2P adatbányász keretrendszer

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

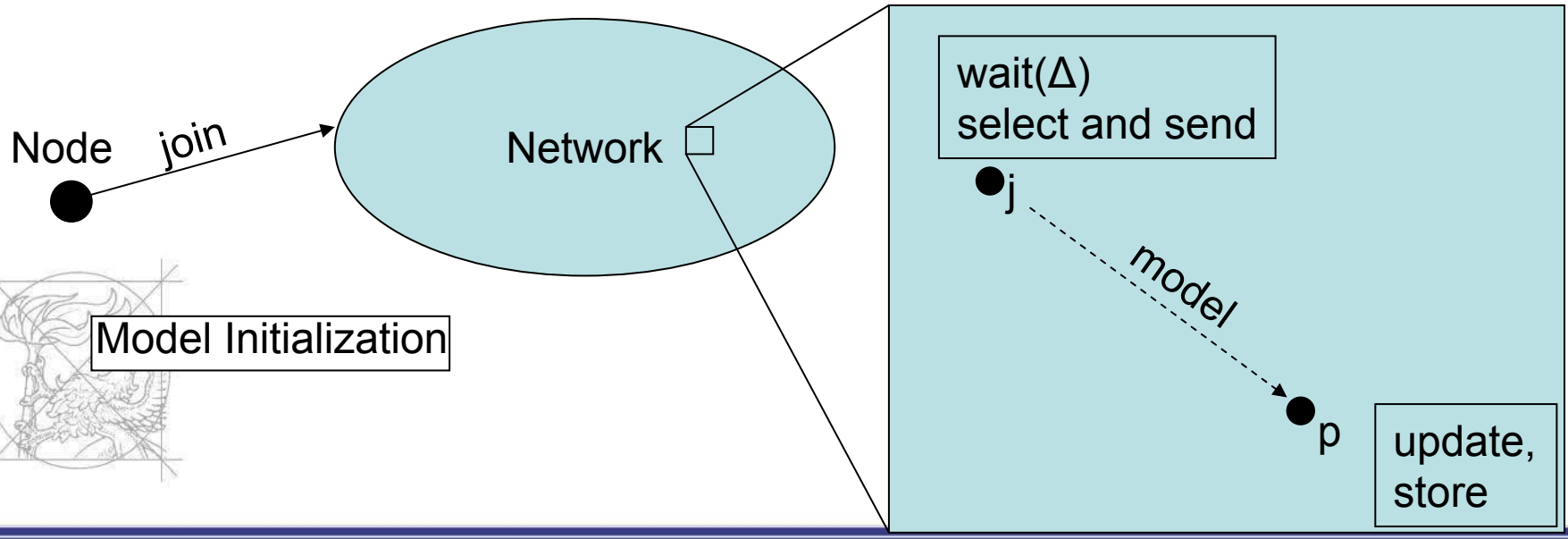
```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p
6: procedure onReceiveModel(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```



P2P adatbányász keretrendszer

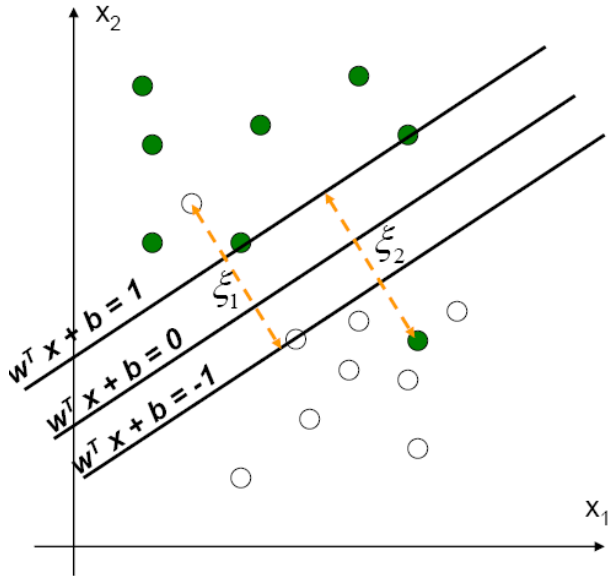
Algorithm 1 P2P Stochastic Gradient Descent Algorithm

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



SGD alapú Support Vector Machines (SVM)

- Pegasos SVM
  - SGD alapú ML algoritmus
  - Egy olyan szeparáló hipersíkot keres (w), amely még maximalizálja a „margót” is



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

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

P2Pegasos SVM algoritmus

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

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

- Definiálnunk kell az updateModel és az initModel metódusokat a különféle ML algoritmusok implementálásához
- Pegasos esetén:

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m
9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

P2Pegasos SVM algoritmus

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p

6: procedure ONRECEIVEMODEL(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```

- Definiálnunk kell az updateModel és az initModel metódusokat a különféle ML algoritmusok implementálásához
- Pegasos esetén:

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m

9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

P2Pegasos SVM algoritmus

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p

6: procedure ONRECEIVEMODEL(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```

- Definiálnunk kell az updateModel és az initModel metódusokat a különféle ML algoritmusok implementálásához
- Pegasos esetén:

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m

9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

P2Pegasos SVM algoritmus

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

```
1: initModel()
2: loop
3:     wait(Δ)
4:     p ← selectPeer()
5:     send currentModel to p

6: procedure ONRECEIVEMODEL(m)
7:     m ← updateModel(m)
8:     currentModel ← m
9:     modelQueue.add(m)
```

- Definiálnunk kell az updateModel és az initModel metódusokat a különféle ML algoritmusok implementálásához
- Pegasos esetén:

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m

9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

P2Pegasos SVM algoritmus

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

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

- Definiálnunk kell az updateModel és az initModel metódusokat a különféle ML algoritmusok implementálásához
- Pegasos esetén:

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m
9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

P2Pegasos SVM algoritmus

Algorithm 1 P2P Stochastic Gradient Descent Algorithm

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

- Definiálnunk kell az updateModel és az initModel metódusokat a különféle ML algoritmusok implementálásához
- Pegasos esetén:

Algorithm 2 P2Pegasos

```
1: procedure UPDATEMODEL(m)
2:     η ← 1/(λ · m.t)
3:     if y ⟨m.w, x⟩ < 1 then
4:         m.w ← (1 − ηλ)m.w + ηyx
5:     else
6:         m.w ← (1 − ηλ)m.w
7:     m.t ← m.t + 1
8:     return m
9: procedure INITMODEL
10:    m.t ← 0
11:    m.w ← (0, . . . , 0)^T
12:    send model(m) to self
```

Predikció – kiértékelés

- Minden node megtartja az utolsó néhány fogadott modellt és azok alapján predikál
  - Egy modell esetén

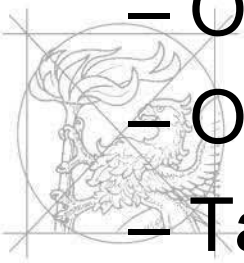
```
1: procedure PREDICT(x)
2:     w ← currentModel
3:     return sign(⟨w, x⟩)
```

  - Több modell esetén (szavaztatás)

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

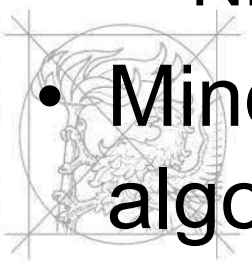
Kommunikációs költség

- A módszer kommunikációs költsége megegyezik a „pletyka” alapú módszerek költségével
  - Minden node a hálózatban Δ időközönként küld egy modellt egy véletlenszerűen választott node-nak
  - O(n) a hálózatban
  - O(1) node-onként
  - Tartalmazza a peer szelekció költségét is



Algoritmus összefoglaló

- Véletlen séta alapú P2P SGD módszer
  - Modell küldés példa „gyűjtögetés” helyett
- Node-ok modelleket gyűjtenek
  - Sima és szavaztatott predikció
- Lokális számítások és kiértékelés
  - Nincs extra kommunikációs költség
- Minden node a hálózatban ugyan azt az algoritmust futtatja



Tesztelés

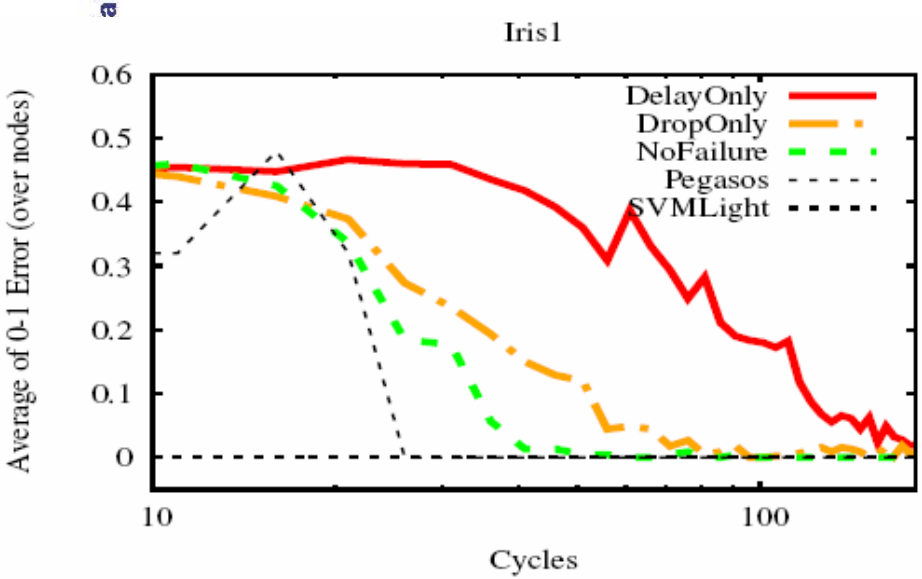
- PeerSim szimulációs környezet
- NewsCast alapú peer szelekció
- A 10 utolsó modell használata szavaztatás esetén
- Baseline-ok: Pegasos, SVMLight
- Adatbázisok: Iris, Reuters, SpamBase, Malicious
- Esetek:
  - Hibamentes (No Failure)
  - Üzenet vesztéses (Drop only): 0.5 valószínűség
  - Késleltetéses (Delay only): [Δ, 10Δ] –ből véletlenszerűen
  - Churn (Churn only): Log-normal eloszlásból
  - Minden együtt (All Failures)

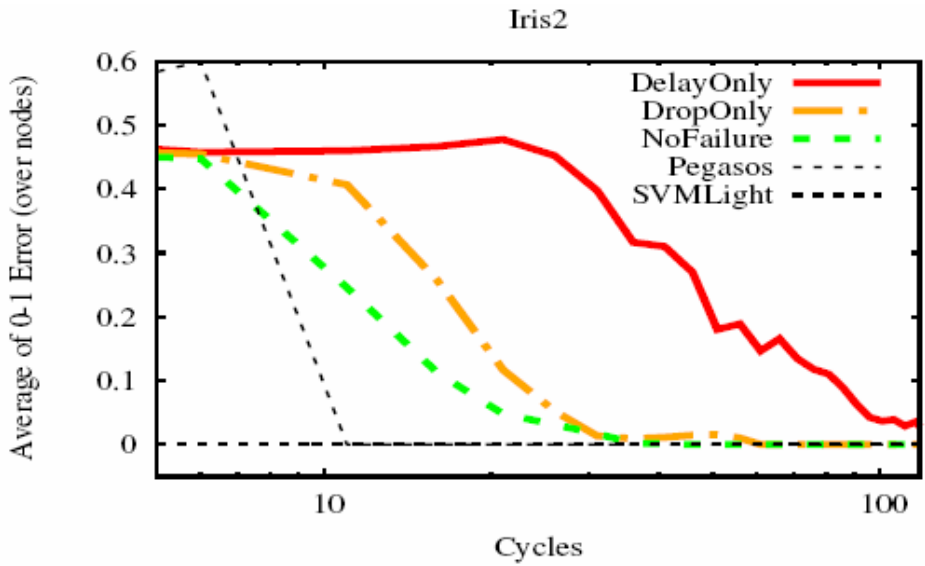
Adatbázis jellemzők

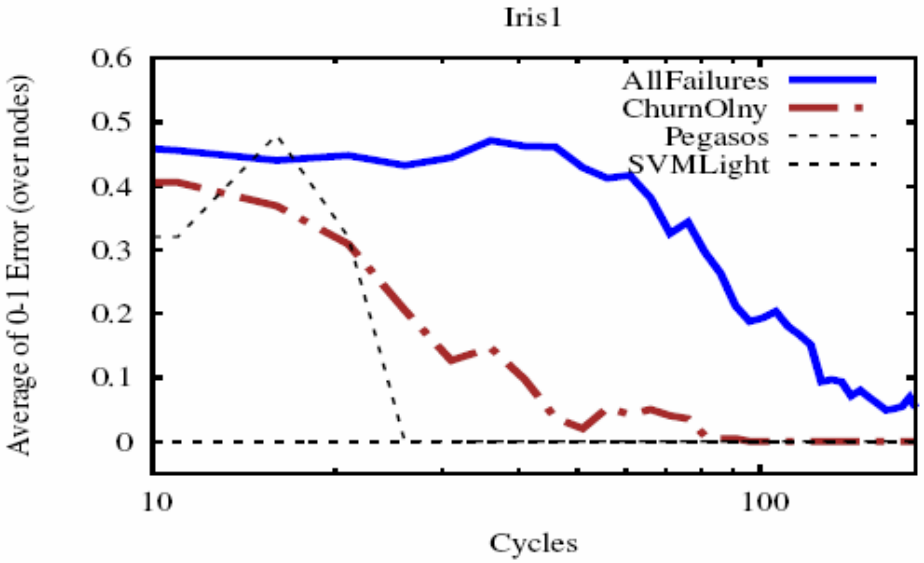
	Iris1	Iris2	Irirs3	Reuters	SpamBase	Malicious10
Training set size	90	90	90	2000	4140	2155622
Test set size	10	10	10	600	461	240508
Number of features	4	4	4	9947	57	10
Classlabel ratio	50/50	50/50	50/50	1300/1300	1813/2788	792145/1603985
Pegasos 20000 iter.	0	0	0	0.025	0.111	0.080 (0.081)
Pegasos 1000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

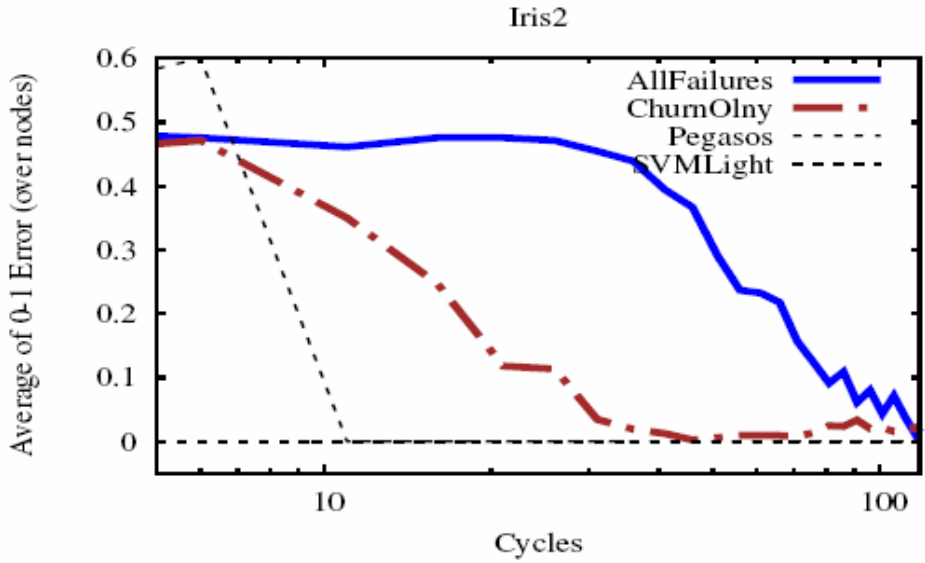
al Intelligence

Hálózati hibák hatásai



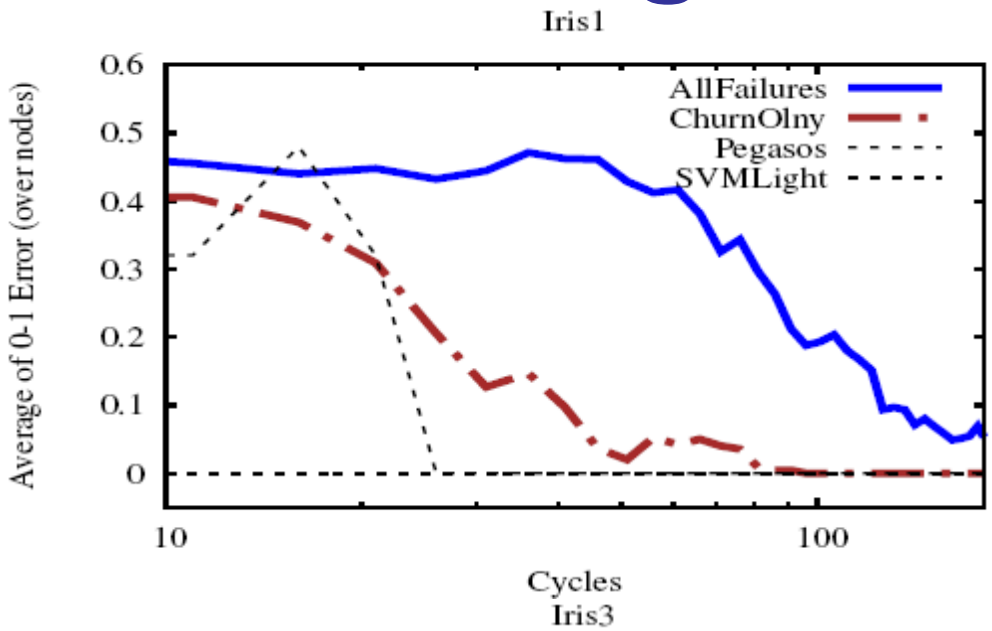


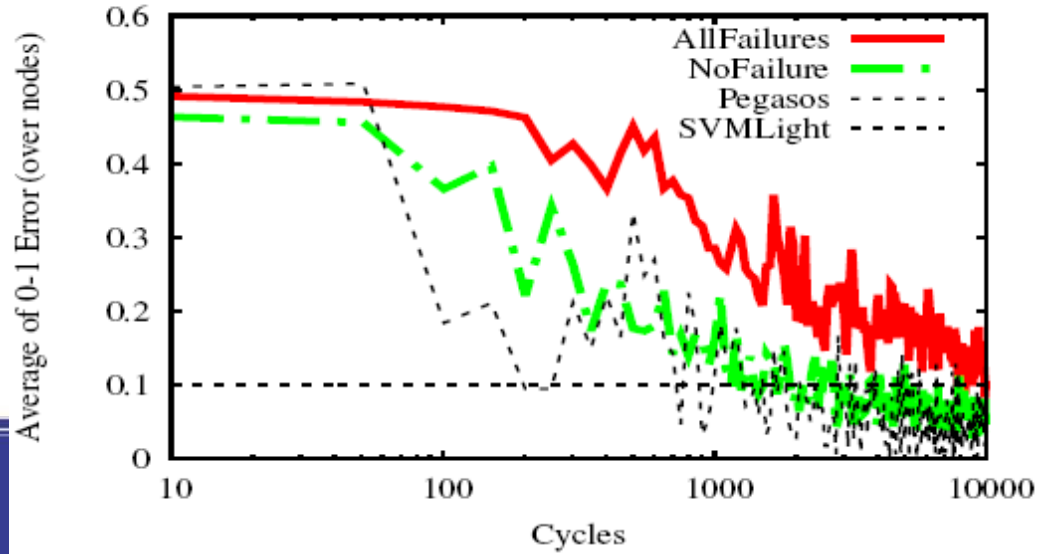




Méret és tanulhatóság

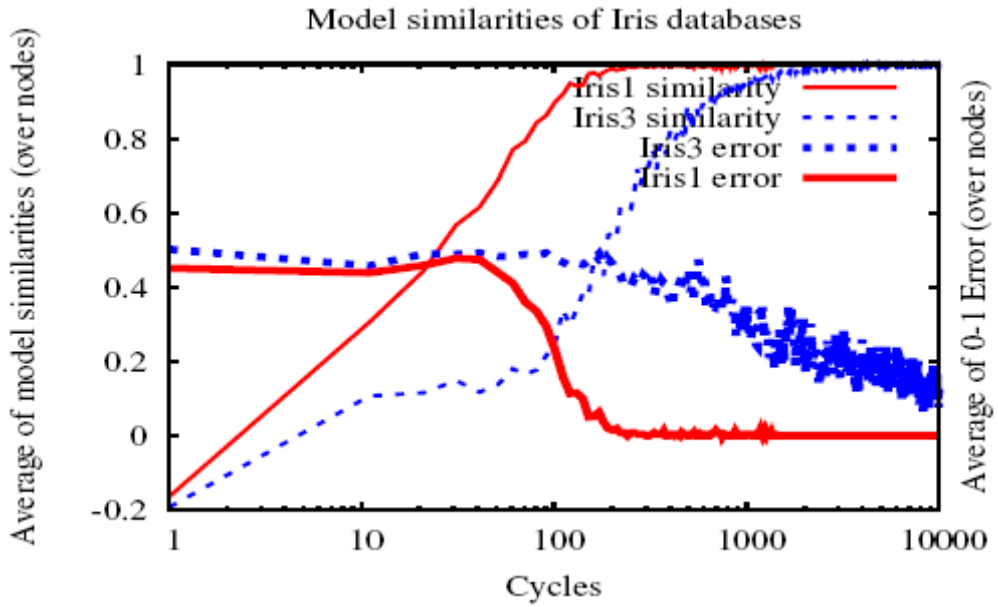
- Nincs összefüggés az adatbázis mérete és tanulhatósága között
- A tanulhatóság inkább az adatbázis struktúrájától függ, mint a méretétől



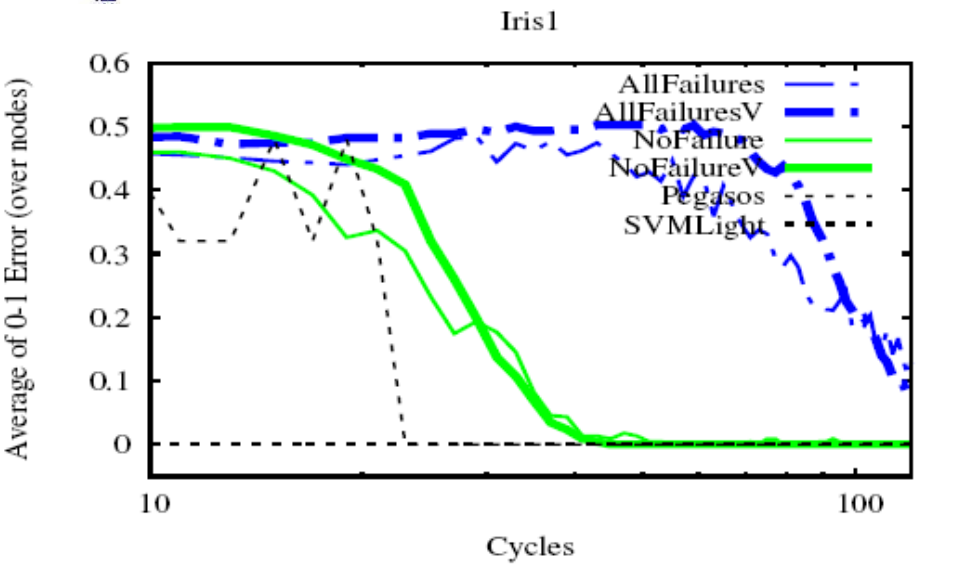


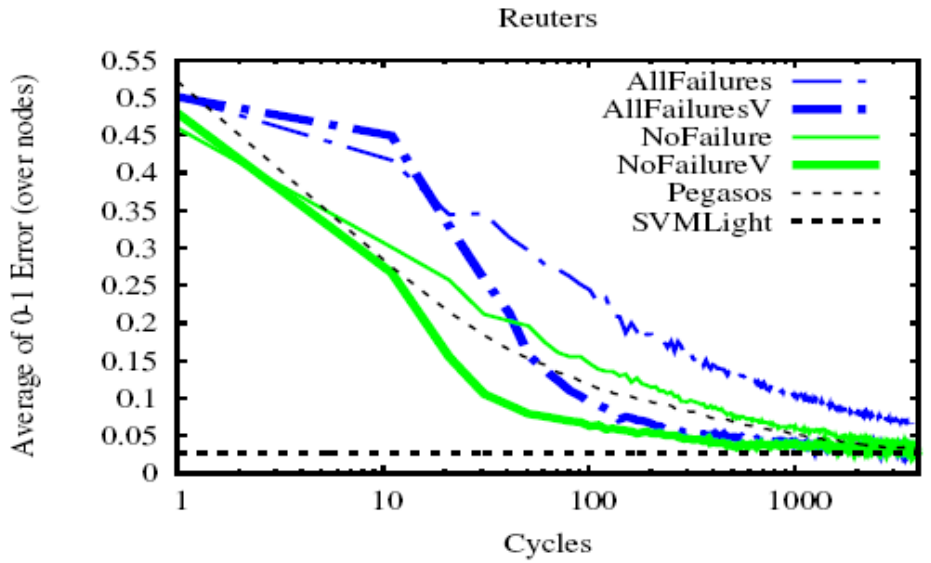
Konvergencia

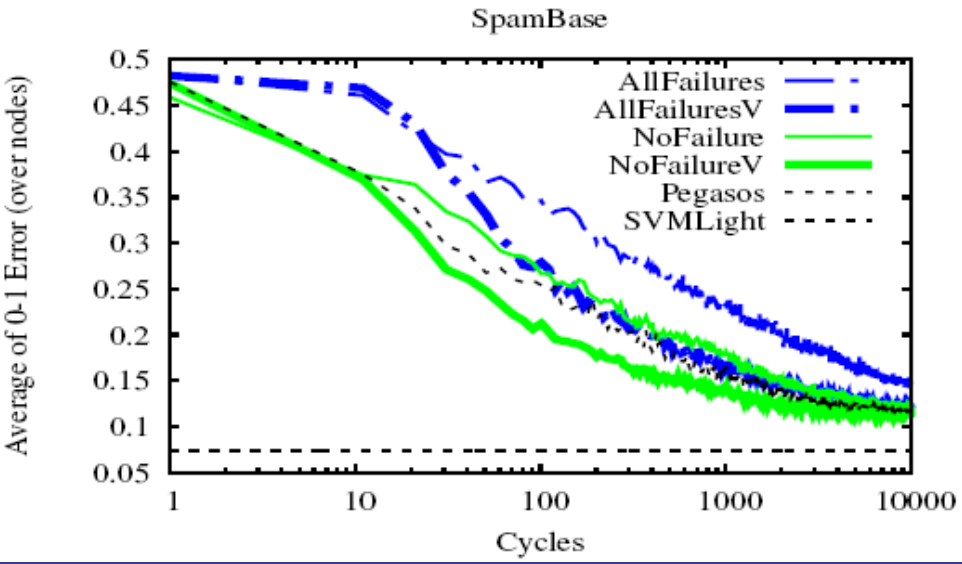
- A teljesítmény és a modell hasonlóság között összefüggés van
- A modell hasonlóság nő a teljesítménnyel
  - mivel a modellek ugyan abba az optimumba konvergálnak

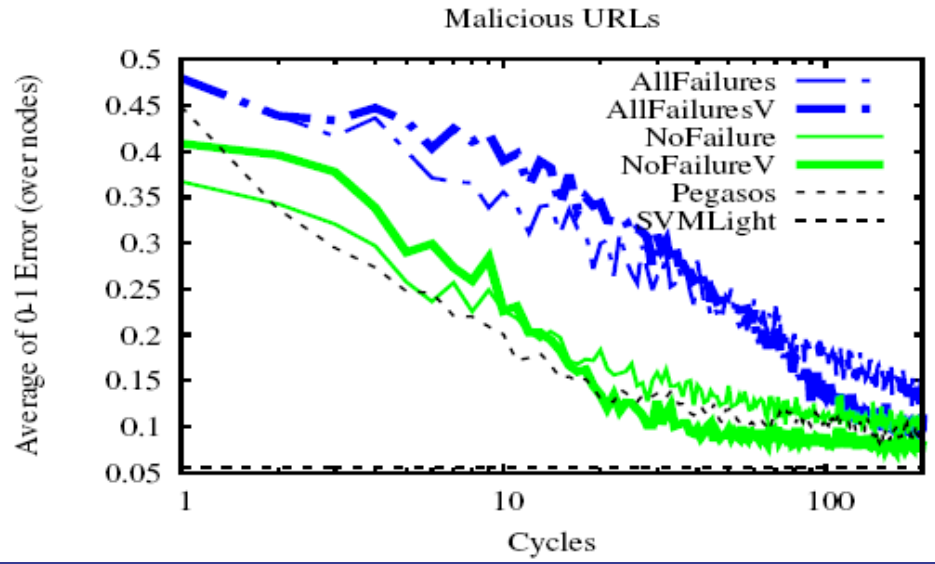


További eredmények









Összefoglaló

- P2P SGD alapú keretrendszer bemutatása
- A keretrendszerbe a Pegasos SVM implementálása valamit az algoritmus tesztelése
- Elvárt teljesítmény sikeres elérése P2P esetben
- Az algoritmus extrém környezetben is megfelelően működik