Lectives

Teljesen elosztott adatbányászat pletyka algoritmusokkal

Jelasity Márk

Ormándi Róbert, Hegedűs István

Motiváció

- Nagyméretű hálózatos elosztott alkalmazások az Interneten egyre fontosabbak
  - Fájlcserélő rendszerek (BitTorrent, stb),
  - Okostelefon alkalmazások
  - Grid, stb
- Ezek számára speciális adatbányász algoritmusokat kell kifejleszteni!
  - Ajánló rendszerek,
  - spam szűrés, stb

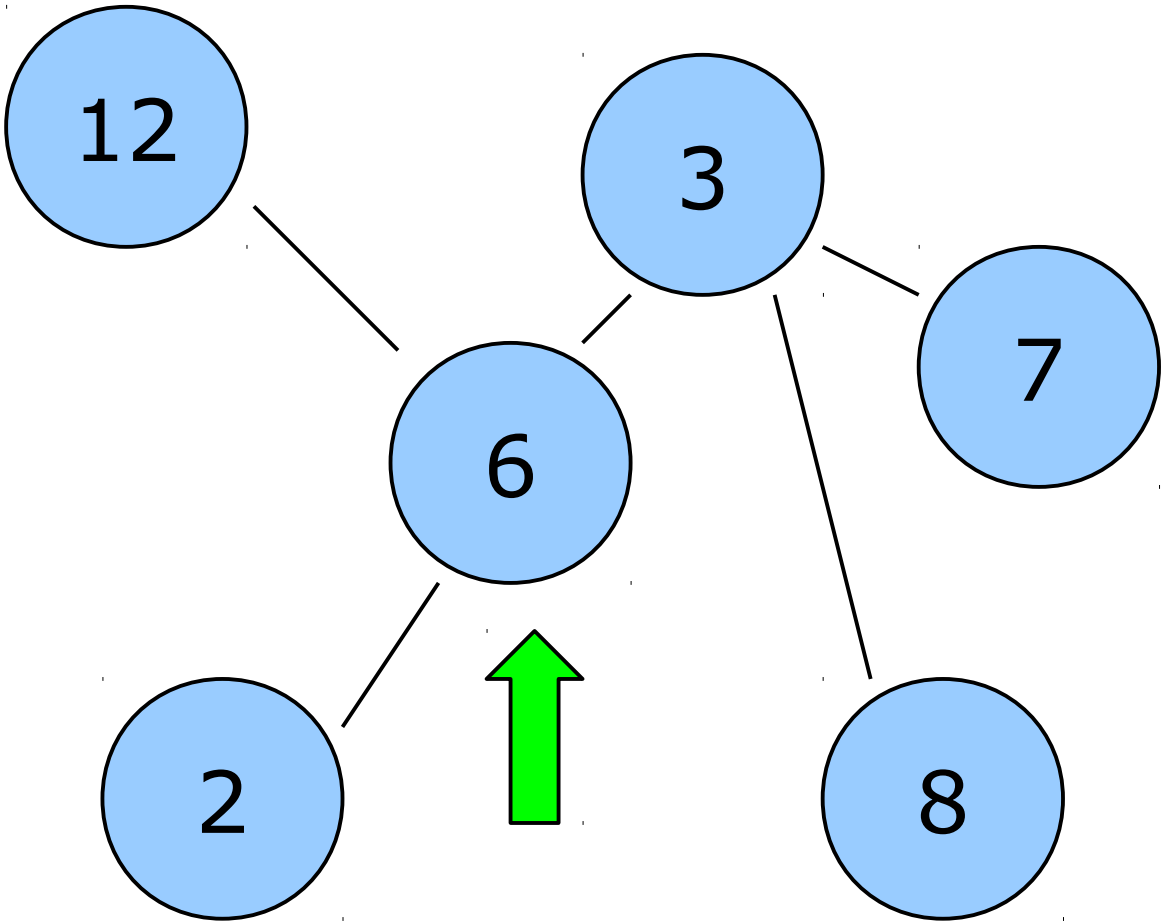
Rendszermodell

- Nagyon sok (több millió) eszköz (számítógép)
  - Ezentúl csomópontnak hívjuk őket (node)
- Csomagkapcsolt hálózat segítségével kommunikálnak
  - Minden csomópont hálózati címmel rendelkezik
  - A cím ismeretében a csomópont számára üzenet küldhető bármely másik csomópontból
- Az üzenetek a hálózatban késhetnek, és el is veszhetnek, sorrendjük sem garantált
- Formalizálhatjuk is, pl. a kaotikus modell érdekes és releváns

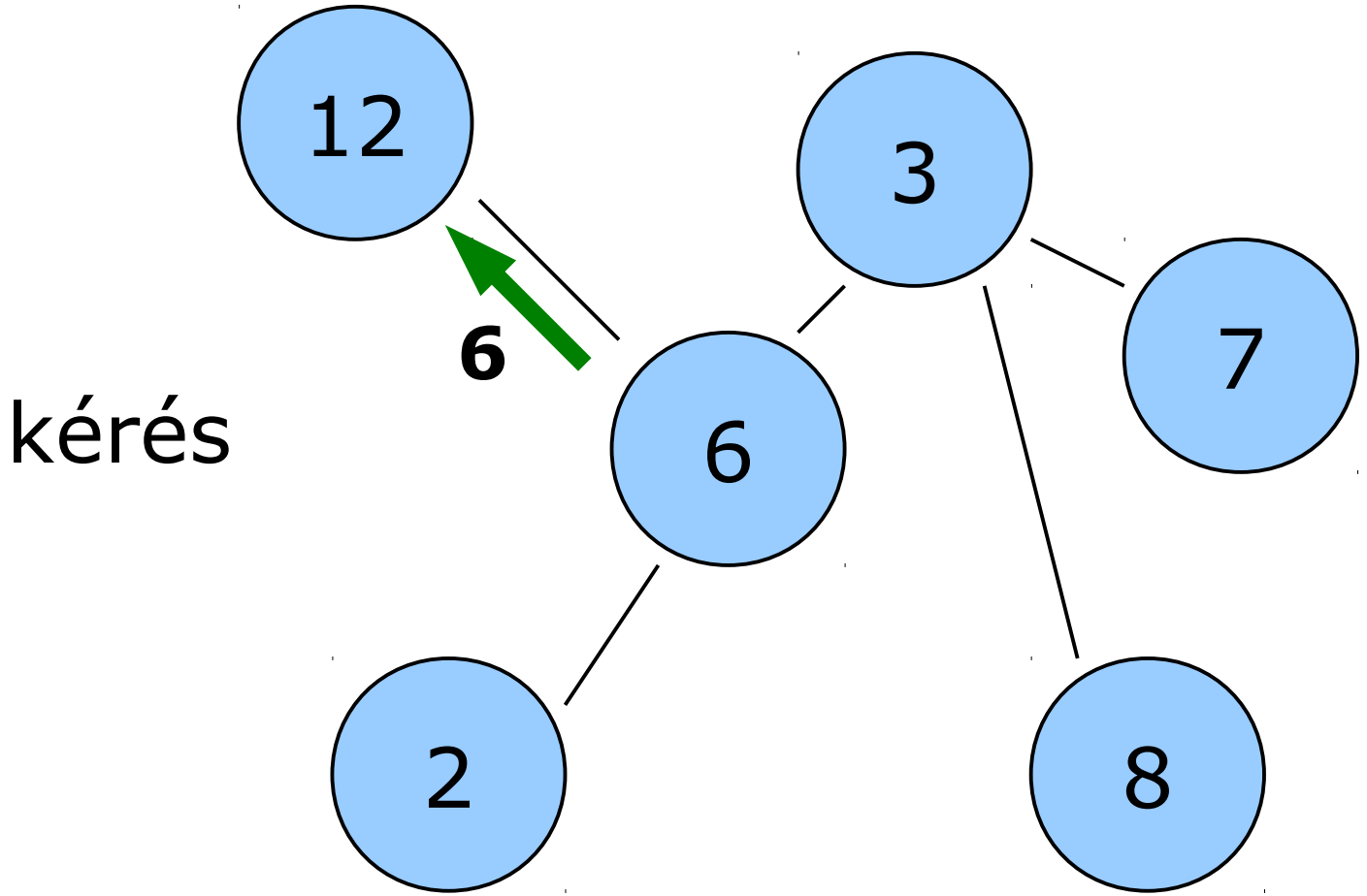
Adatmodell

- Minden csomóponton kevés, esetleg csak egy adatrekord (innen feltesszük hogy pont egy)
- Nem engedjük meg az adat mozgatását, csak lokális feldolgozást
  - Privacy preservation (magánélet tisztelete)
- Az előállított modellek használata olcsó és minden csomópont számára elérhető legyen
  - Demokratikus feltétel

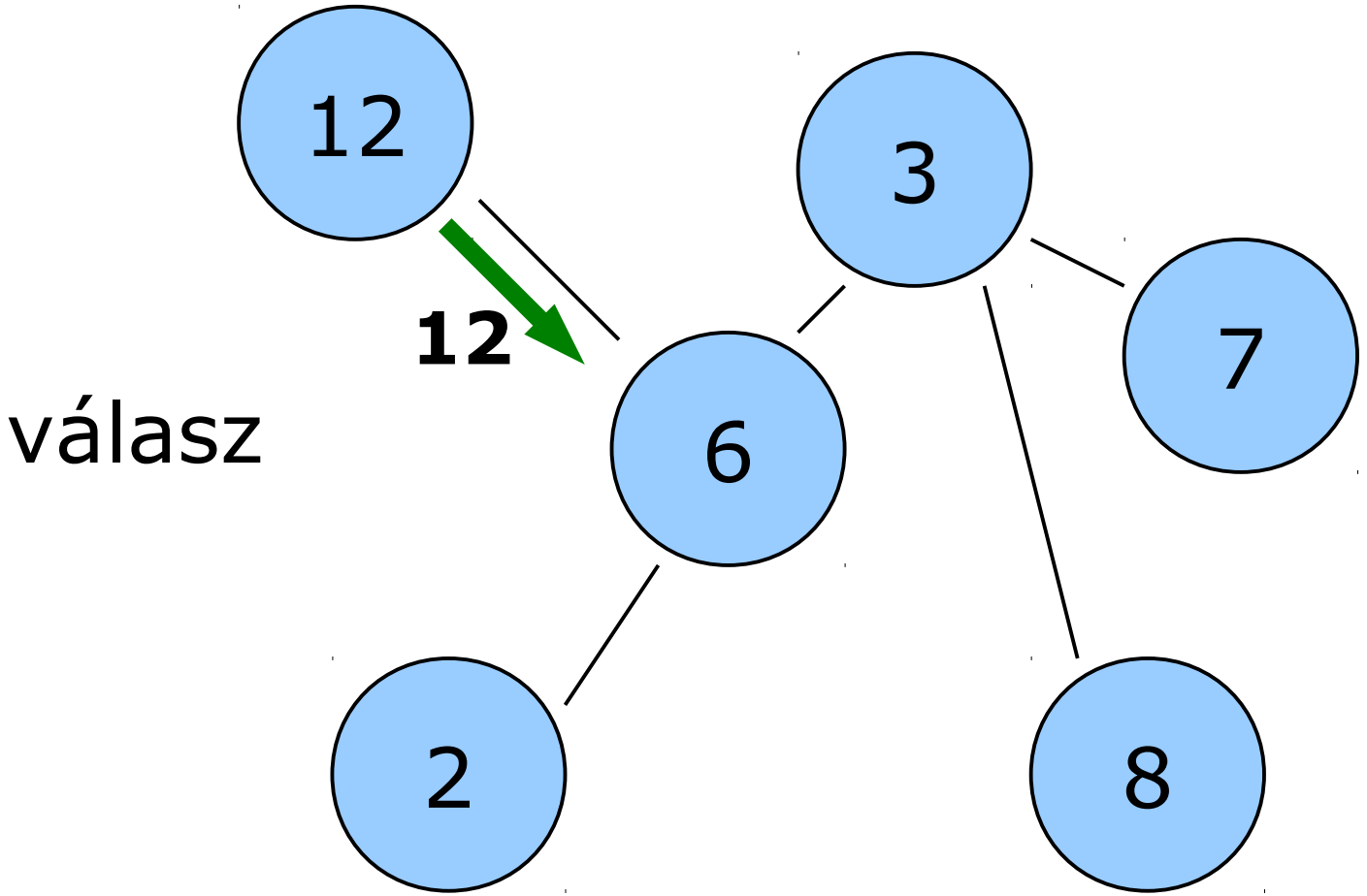
Illusztráció: átlagolás



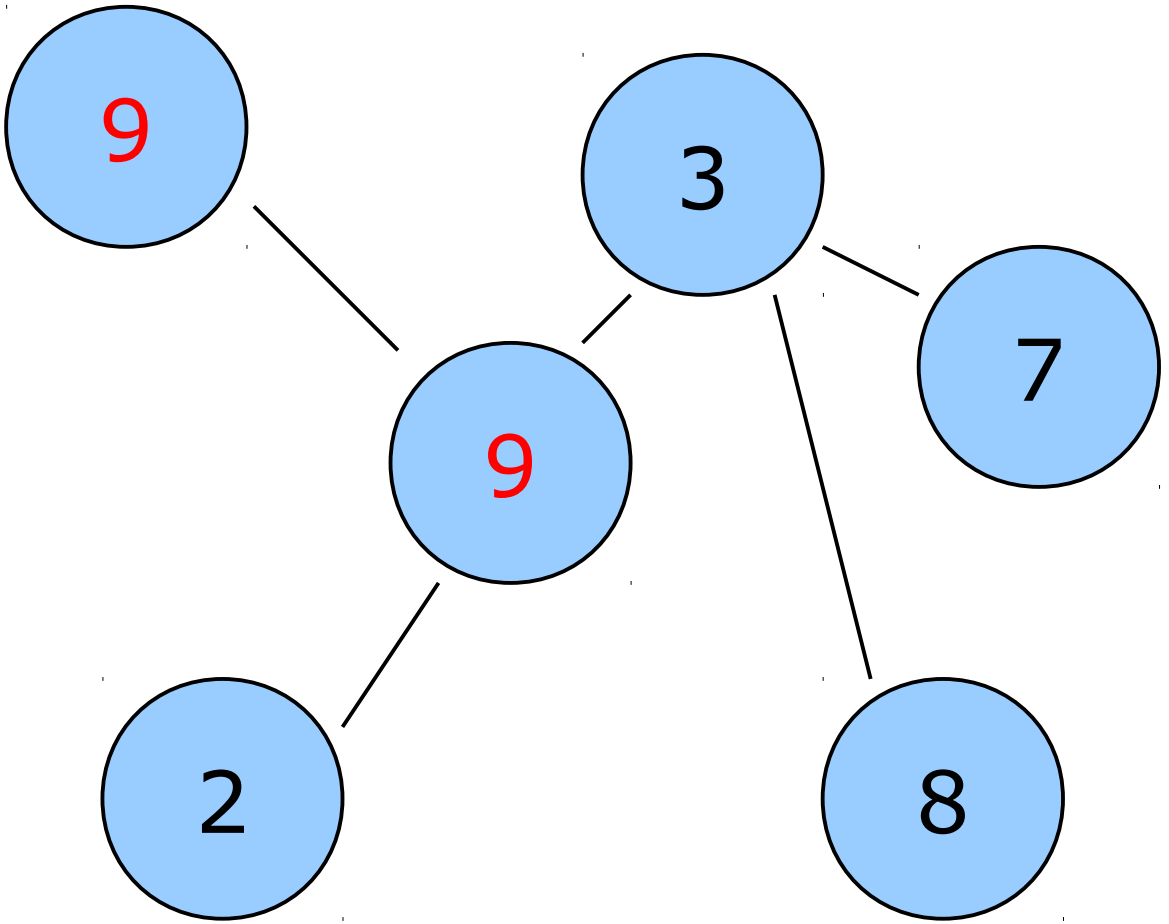
Illusztráció: átlagolás



Illusztráció: átlagolás



Illusztráció: átlagolás

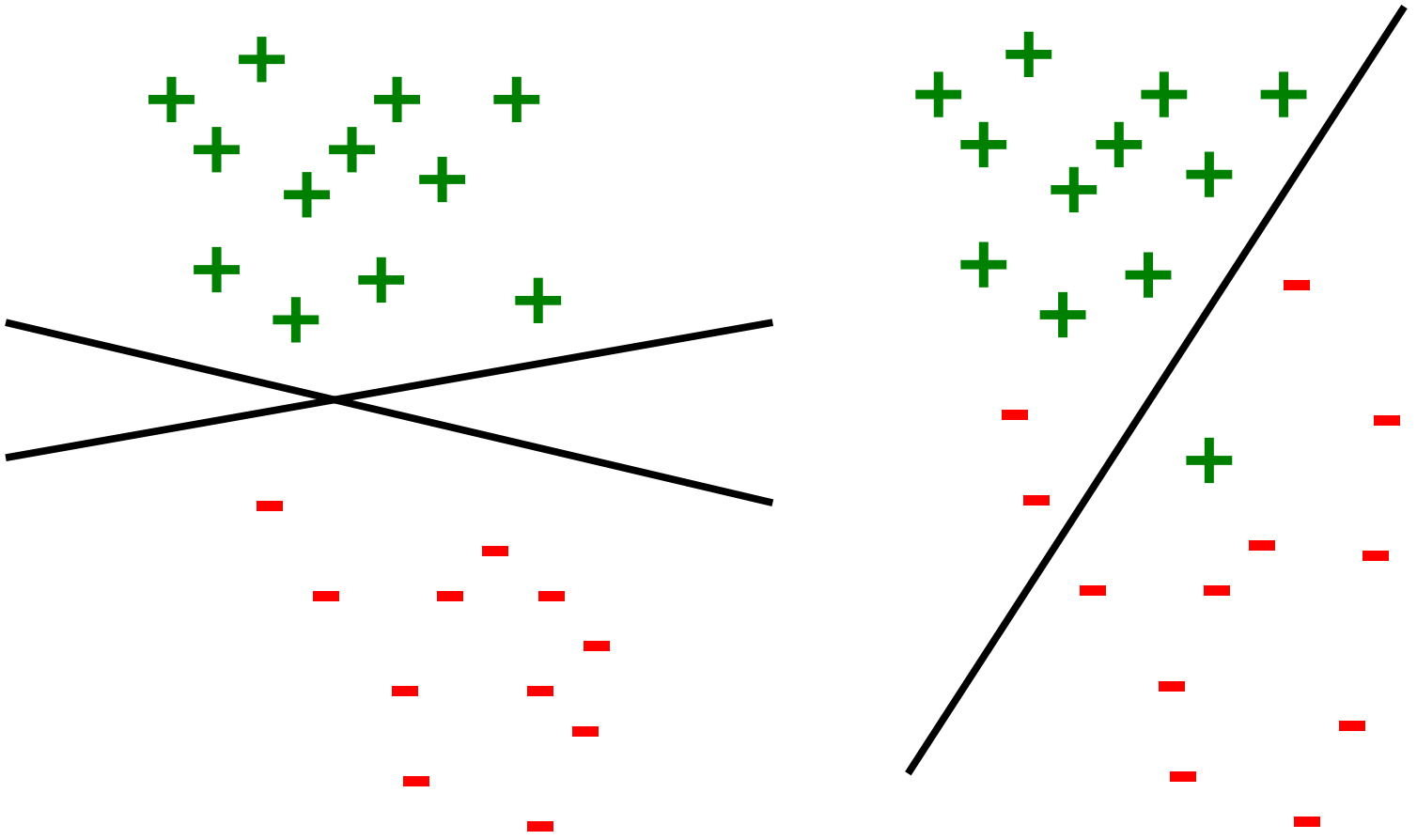


(12+6)/2=9

Osztályozási probléma

- Adott (x_i,y_i) példák egy halmaza, ahol y_i x_i osztálya (y_i pl. -1 vagy 1, kétosztályos esetben)
- Egy f() modellt szeretnénk, amelyre $f(x_i)=y_i$ (ill. $f(x_i)\approx y_i$) minden i-re
- f() gyakran paraméterekkel addott: $f_w()$, így a tanulási probléma hibaminimalizálásra vezethető vissza w-ben.
- A hiba gyakran a példák feletti hibák összege

Osztályozás lineáris modellel



Teljesen elosztott osztályozás

- A probléma tehát olyan optimalizáló algoritmust találni, amely jól illeszkedik a rendszer- és adatmodellünkbe
- A legtöbb ismert módszer erősen szinkronizált, és felteszi az olcsó véletlen hozzáférést a teljes adatbázishoz
- Az online módszerek kivételt képeznek!
  - Egyszerre csak egy adatrekordhoz férnek hozzá
  - Ezzel a rekorddal frissítik a modellt
- A sztochasztikus gradiens módszer egy gyakori online módszer, ezt alkalmazzuk a lineáris modellekre (az SVM módszer hibafüggvényének primál alakjára)

Sztochasztikus gradiens

- Tegyük fel hogy a hiba:
- Ennek gradiense:
- Tehát a teljes gradiens módszer az lenne hogy:
- De csak egy példát veszünk egyszerre, szóval a módszer:

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

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

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

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

A pletyka tanulás

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
```

A „merge” függvény

- Legyen z=merge(x,y)=(x+y)/2 (x, y lineáris modellek)
- Adaline perceptron stochasztikus gradiens módszerével
  - z frissítése egy példával u.olyan hatású mint x és y frissítése az adott példával, majd ezek átlagolása
  - z-vel predikálni u.az, mint x és y predikcióját súlyozottan átlagolni
- Ez azt jeleni, hogy effektíve egy exponenciálisan növekvő számú modellt propagálunk, és ezek szavaztatása a predikciónk!
- Az lineáris SVM algoritmusra ez nem teljesül pontosan, de a hasonlóság heurisztikusan motiválja a módszert

Lokális predikció

- Csak helyben meglévő modelleket használunk
  - Vagy csak az aktuális modellt
  - Vagy az „ingyen” összegyűlt modellek szavaztatását is elvégezhetjük

```
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
```

Kísérleti kiértékelés

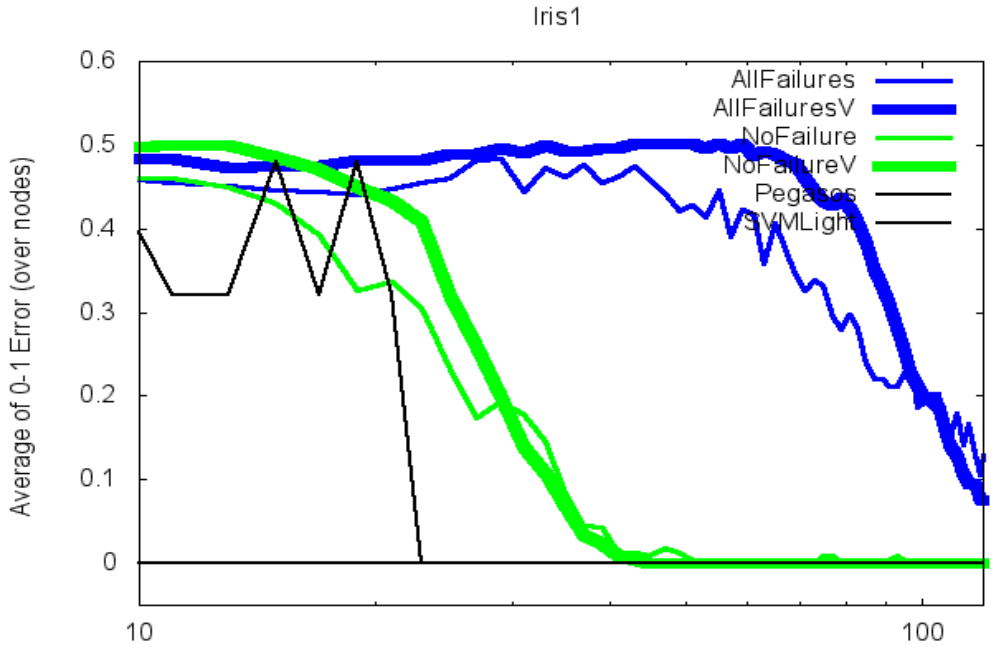
- Az ismert Pegasos algoritmust alkalmaztuk (lineáris SVM sztochasztikus gradiens módszerrel) a pletyka keretben
- Ismert adatbázisokon értékeltük ki a módszert
  - Teljesen elosztott adatmodell, egy rekord egy csomóponton
- Extrém kísérleti beállításokat is használtunk a robosztusság tesztelésére (kaotikus modell)
  - 50% üzenetvesztés
  - 1-10 ciklusnyi késleltetés

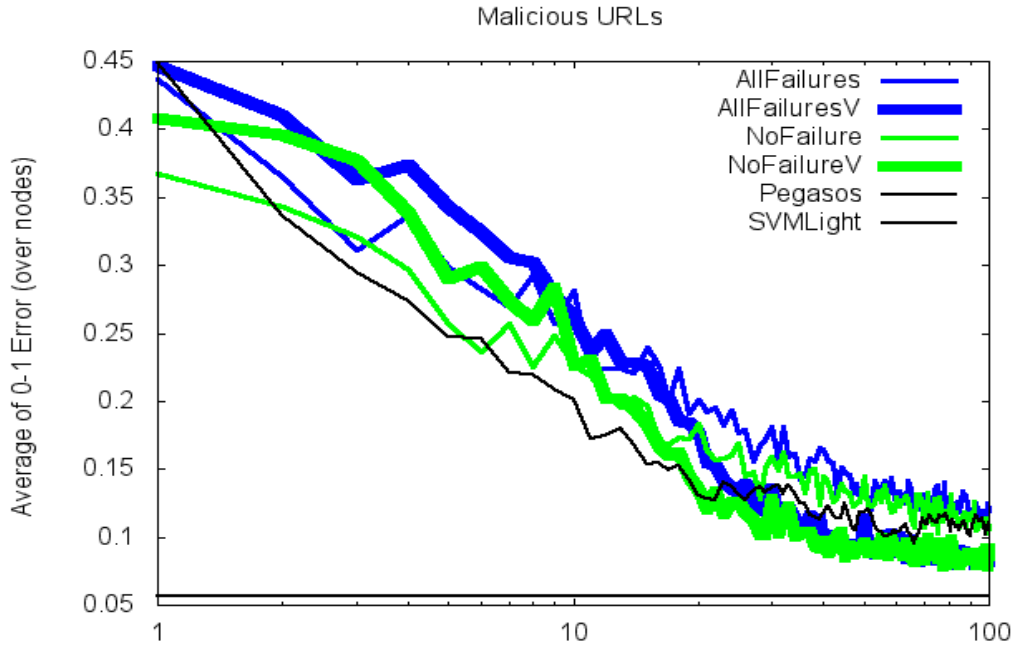
Tanuló adatbázisok

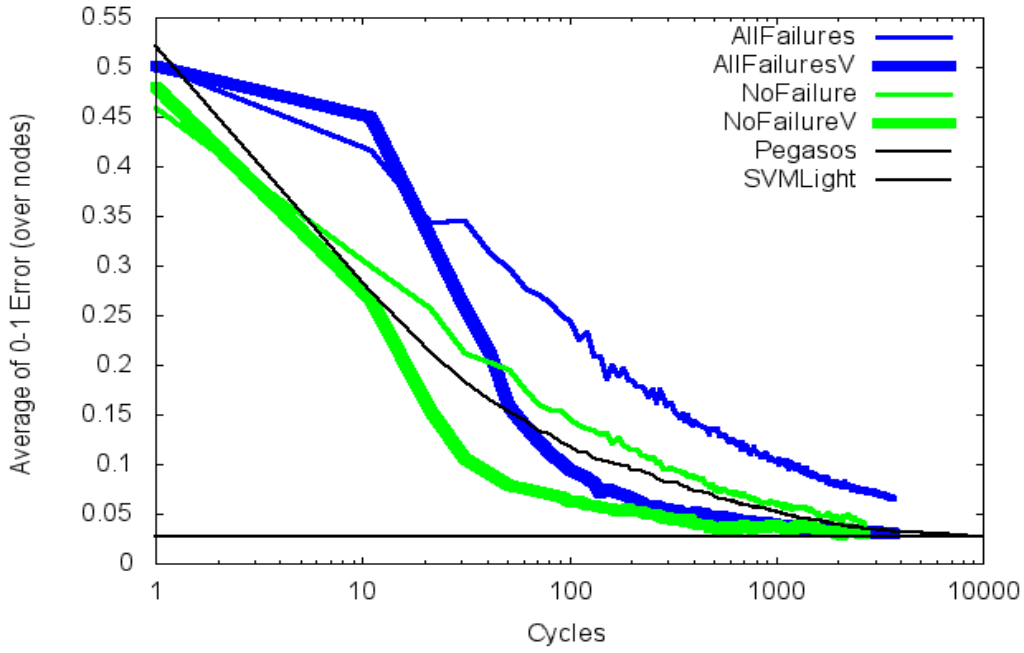
	Iris1	Iris2	Irirs3	Reuters	SpamBase	Malicious10
Training set size	90	90	90	2000	4140	2155622
Test set size	10	10	10	600	461	240508
Number of features	4	4	4	9947	57	10
Classlabel ratio	50/50	50/50	50/50	1300/1300	1813/2788	792145/1603985
Pegasos 20000 iter.	0	0	0	0.025	0.111	0.080 (0.081)
Pegasos 1000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

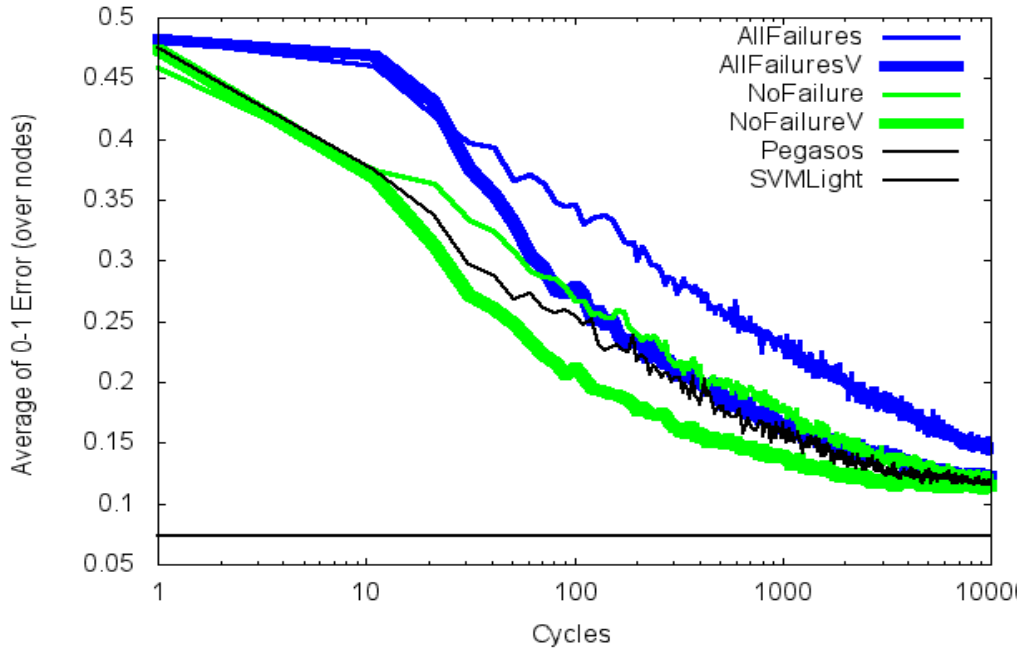
- A kísérletekben felhasznált adatbázisok statisztikái
- Néhány ismert algoritmus teljesítménye

Véletlen séta alapú módszer

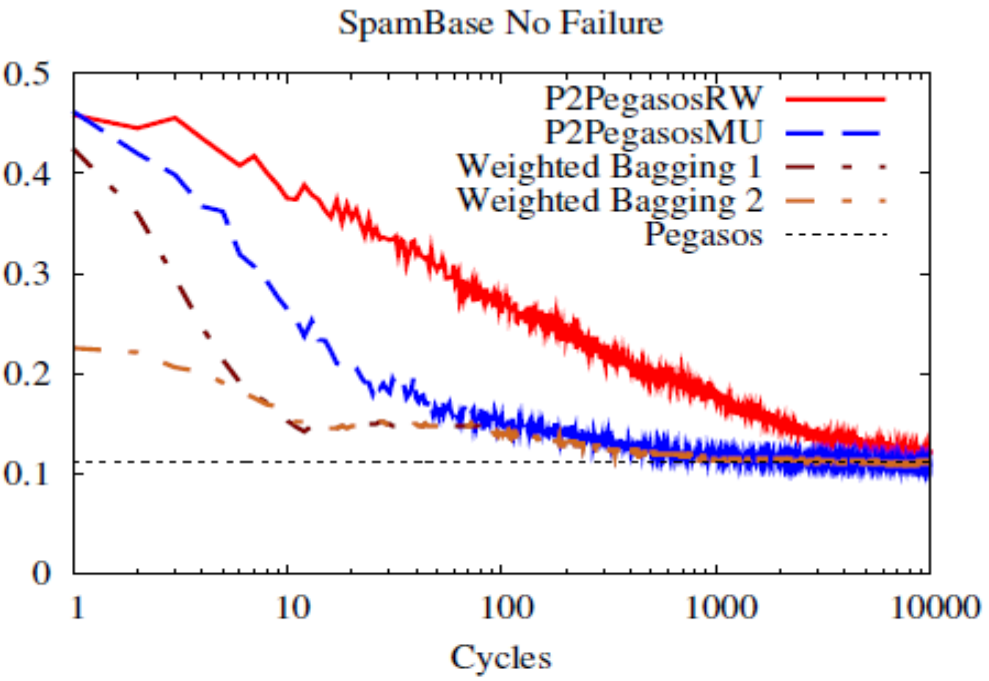


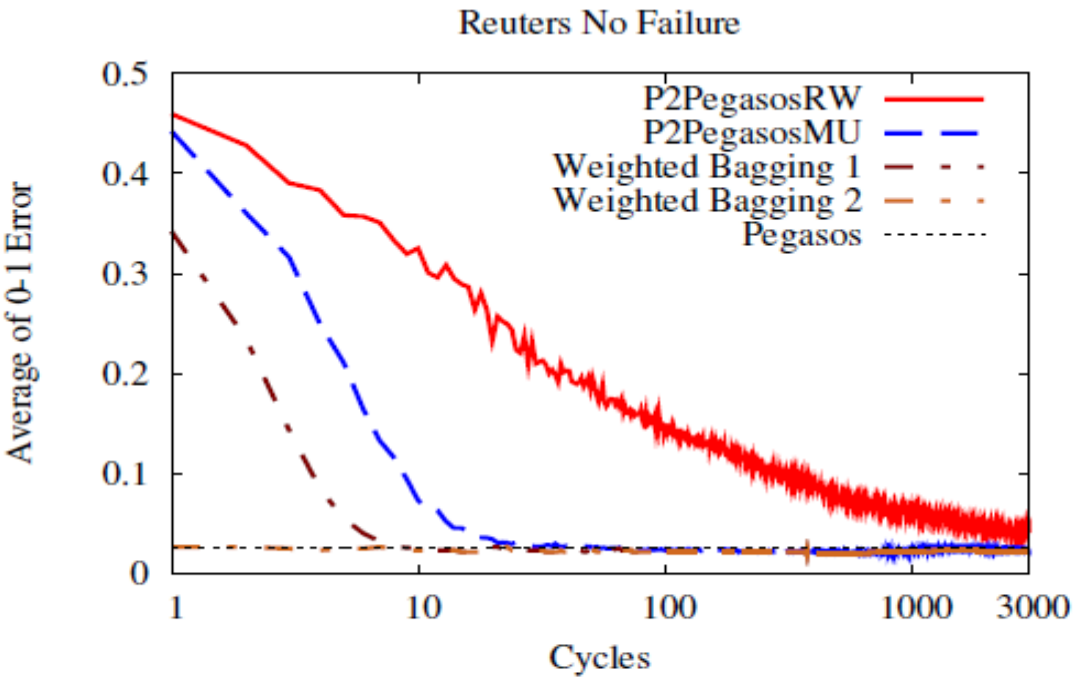


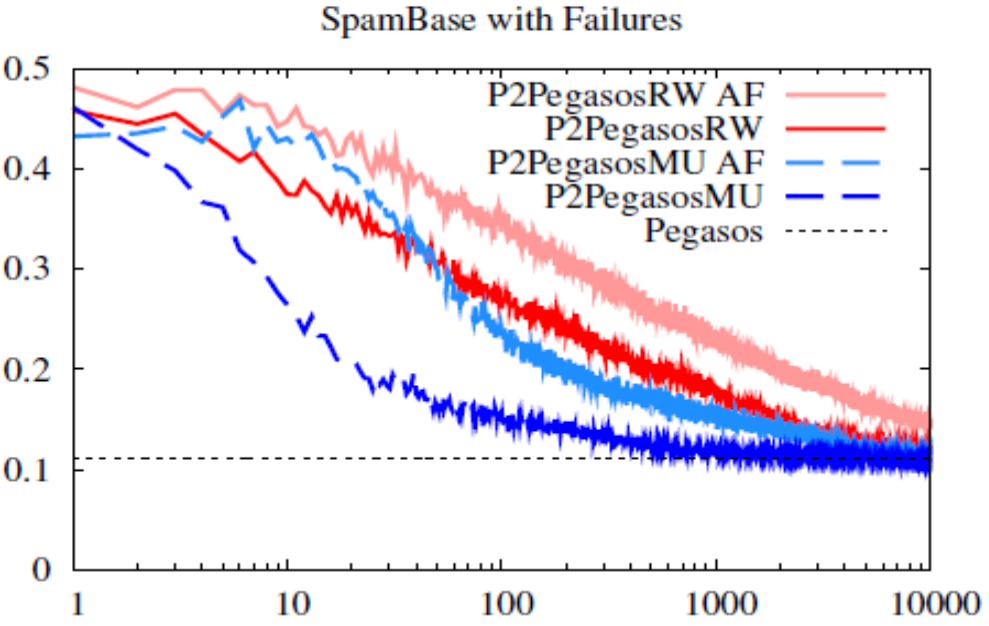


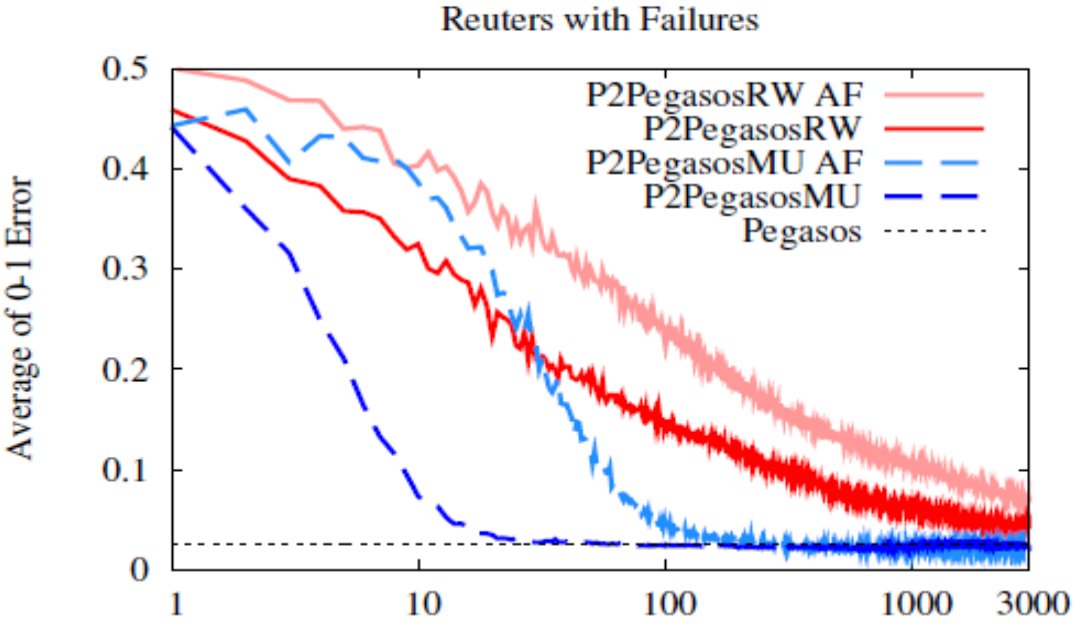


Átlagolt modellek









Megjegyzések

- Ha a megvalósított véletlen séta garantáltan uniform, akkor az összes modell bizonyíthatóan az optimális modellhez konvergál
- Ha a késleltetés és üzenetvesztés statisztikailag független a csomópontoktól, akkor függetlenül a késleltetésektől és az üzenetvesztés mértékétől a véletlen séta továbbra is uniform marad
- A gyakorlatban természetesen nem kritikus az uniformitás, de ez függ az adatbázis komplexitásától, a hálózat méretétől, és a „bias” természetétől is