

Szegedi Tudományegyetem
Informatikai Tanszékcsoport

**Momentum módszer a teljesen elosztott gépi
tanulásban**

Diplomamunka

Készítette:
Danner Gábor
Programtervező informatikus
MSc hallgató

Témavezető:
Dr. Jelasity Márk
Tudományos főmunkatárs

Szeged
2014

Tartalomjegyzék

Feladatkiírás	3
Tartalmi összefoglaló	4
1. A pletyka tanulás és a GoLF	5
1.1. Fogalmi bevezető	5
1.2. A pletyka tanulás	5
1.3. A Gossip Learning Framework	6
1.4. Tesztelés	7
2. Súlyozott összeolvasztás	8
3. Momentum	11
4. Összeolvasztható momentum	12
5. Concept drift	17
5.1. Concept drift a Gossip Learning Frameworkben	17
6. Összeolvasztható AdaGoLF	18
7. Modellcsomag módszer	20
8. Forgó csomag módszer	22
9. Momentum és concept drift	24
Nyilatkozat	26
Köszönetnyilvánítás	27
Irodalomjegyzék	28

Feladatkiírás

Az adatbányászatban egyre gyakrabban merül fel, hogy teljesen elosztott, nem egy központi helyen levő nagyméretű adathalmaz felett kell tanuló algoritmusokat alkalmazni. A feladat ilyen algoritmusok implementációja (szimulációban) és elemzése.

Tartalmi összefoglaló

A Gossip Learning Framework keretrendszer segítségével teljesen elosztott gépi tanuló algoritmusok módosításait dolgoztam ki és teszteltem. Sikeresen alkalmaztam egy módosított momentum módszert, mellyel javult a konvergencia sebessége, valamint összeolvasztást tettem lehetővé concept driftes környezetben.

Kulcsszavak: felügyelt tanulás, gradiens módszer, momentum, pletyka tanulás, concept drift.

1. fejezet

A pletyka tanulás és a GoLF

1.1. Fogalmi bevezető

A felügyelt tanulás egy leképezésnek a bemeneti és kimeneti minták alapján történő megtanulását jelenti.[9] Bináris kimenettel rendelkező tanulási problémát bináris osztályozási feladatnak hívjuk.[2] Egy lineáris osztályozó a jellemzők lineáris kombinációjának értéke alapján hozza meg a döntést.[1] A dolgozatomban vizsgált bináris lineáris osztályozási módszerek a logisztikus regresszió[1] és a szupport vektor gépek (SVM)[2].

Ezen modellek tanítása visszavezethető egy optimalizálási feladatra, ahol a célfüggvény olyan deriválható hibafüggvények összege, melyek egy-egy tanító példához tartoznak. Ez megoldható pl. gradiens módszerrel, azaz a modell iteratív módosításával, mely a célfüggvény deriváltja és egy folyamatosan csökkenő lépésköz alapján történik. Batch tanulásról akkor beszélünk, ha minden iterációs lépésben kiszámítjuk a teljes gradienst. Ez azonban nagy tanító adatbázisnál igen költséges. Így elterjedtebb a sztochasztikus gradiens módszer, mely minden iterációs lépésben egy mintavételezést hajt végre a tanító adatbázison, és csak ezen részhalmaz alapján közelíti a gradienst.[3] Ha ezek a részhalmazok egy eleműek, online tanulásról beszélünk, ha több eleműek, akkor mini-batch tanulásról. A mini-batch hatékonyabb lehet, mert jobban közelíti az eredeti gradienst. A mini-batch-hez hasonló technika a momentum, mely az eddig látott példákhoz tartozó gradiensek csúszoátlagára alapján módosítja a modellt.[5]

1.2. A pletyka tanulás

Egy külön kihívást jelent teljesen elosztott adatokon, peer-to-peer hálózaton hatékonyan működő tanuló algoritmusok kifejlesztése, különösen, ha a tanító adatok nem hagyhatják el az eszközöket, a személyes adatok védelme érdekében. A kommunikáció nem megbízható, és az egyes eszközökön tárolt tanító példák száma lehet mindösszesen egy is. Erre a problémára kínál megoldást a pletyka tanulás[8], melynek alapötlete, hogy párhuzamosan több modell véletlenszerűen sétál a hálózaton, és eközben online tanulnak a lokálisan elérhető adaton, valamint kombinálódnak egymással. A folyamatos kombinációnak köszönhetően exponenciálisan sok modelltől gyűjthető információ.

A pletyka tanulás sémája¹ [6] az alábbi:

¹ Az egyszerűség kedvéért modell cache nélkül.

```
procedure activeLoop()
  initModel()
  loop
    wait( $\Delta$ )
     $p \leftarrow \text{selectPeer}()$ 
    send currentModel to  $p$ 
  end loop
end procedure

procedure onReceiveModel( $m$ )
  currentModel  $\leftarrow$  createModel( $m$ , lastModel)
  lastModel  $\leftarrow m$ 
end procedure
```

Ez az algoritmus fut a hálózat minden eszközén. A *selectPeer()* egy overlay hálózati protokoll (pl. NewsCast) által biztosított szolgáltatás, mely egy véletlenszerűen választott peert ad vissza. Az eszköz körülbelül Δ időközönként elküldi az aktuális modelljét egy másiknak. Amikor érkezik egy modell, létrehozunk egy új modellt az utolsó két beérkezett modell és a lokális tanító példák alapján. Ez válik az új aktuális modellé, amit predikcióhoz is használunk.

Az absztrakt *createModel($m1$, $m2$)* függvény többféleképp is implementálható:

```
procedure createModelRW( $m1$ ,  $m2$ )
  return update( $m1$ )
end procedure

procedure createModelMU( $m1$ ,  $m2$ )
  return update(merge( $m1$ ,  $m2$ ))
end procedure
```

A *createModelRW($m1$, $m2$)* véletlen sétát valósít meg, valójában nem olvaszt össze modelleket. A *createModelMU($m1$, $m2$)* összeolvasztja a két modellt, utána pedig az így létrejövő modellt tanítja. Lineáris modellek esetén ez az összeolvasztás megvalósítható a modellek átlagolásával.[6]

A pletyka tanulás robusztus², és az összeolvasztás alkalmazása jelentősen megnöveli a konvergencia sebességét.[6]

1.3. A Gossip Learning Framework

A Gossip Learning Framework (GoLF)[7] egy nyílt forráskódú keretrendszer peer-to-peer tanuló algoritmusok kiértékelésére. Az általa szimulált hálózatos környezet képes modellezni a kommunikáció megbízhatatlanságait. A GoLF tartalmazza továbbá több ismert gépi tanuló algoritmus implementációját.

² A következő alfejezetben felsorolt hibákra nézve.

A szimulálható hibák a következők:

- az üzenetek késve érkezhetnek meg;
- az üzenetek elveszhetnek;
- az eszközök "elaludhatnak" egy időre, mely alatt nem tudnak csomagot küldeni vagy fogadni, vagy bármi egyebet csinálni.

A keretrendszer többek között az alábbi implementációkat tartalmazza:

- *LogisticRegression*: Logisztikus regresszió összeolvasztás nélkül
- *P2Pegasos*: Pegasos SVM összeolvasztás nélkül
- *MergeableLogisticRegression*: Logisztikus regresszió összeolvasztással
- *MergeablePegasos*: Pegasos SVM összeolvasztással

A modellek tartalmaznak egy "életkort", ami minden tanító példán való tanulás után megnő eggyel. A lépésköz fordítottan arányos az életkorral. Összeolvasztáskor a két modell életkorának maximuma lesz az új modell életkora.

1.4. Tesztelés

A concept drift előtti fejezetekben a spambase adatbázist használok az algoritmusok kiértékelésére. A tanító halmaz 4140 példát tartalmaz, ebből 1620 (39%) pozitív. A teszt halmaz 461 példát tartalmaz, ebből 193 (42%) pozitív. Minden tanító példához létrejön egy peer a hálózatban. A grafikonokon a teszt halmazon mért predikciós hiba szerepel (az egyes peerek modelljei fölött átlagolva), az idő függvényében.

2. fejezet

Súlyozott összeolvastás

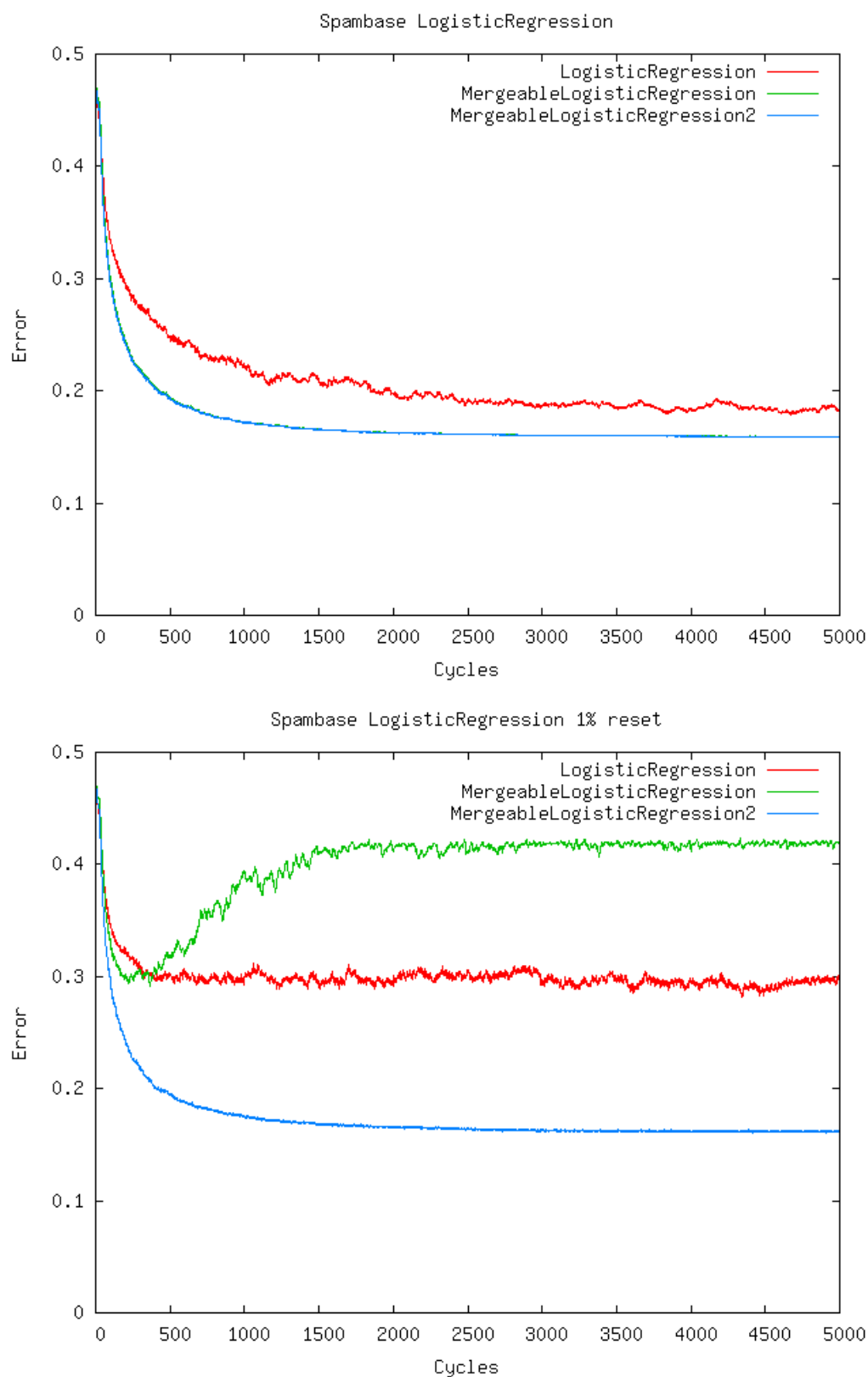
Ugyan bizonyos fajta lemorzsolódás szimulációját tartalmazta már a keretrendszer, de aszerint az eszközök csak ideiglenesen hagyják el a hálózatot, és később az elhagyáskori állapotukkal térnek vissza. Viszont kíváncsi voltam, mi történik, ha egyes eszközök véglegesen elhagyják a hálózatot, valamint utólag csatlakoznak be újak, avagy hogy mekkora az algoritmus robusztussága a peerek cserélődésével szemben.

Az egyszerűség kedvéért ezt úgy szimuláltam, hogy a módosított protokoll az aktív ciklus végén előre definiált valószínűséggel "resetelhesse" a tárolt modelleket, azaz újrainicializálhassa őket. Egy ilyen esemény egy peer-cserét jelképez, azaz egy kilépő és egy belépő eszköznek felel meg.¹

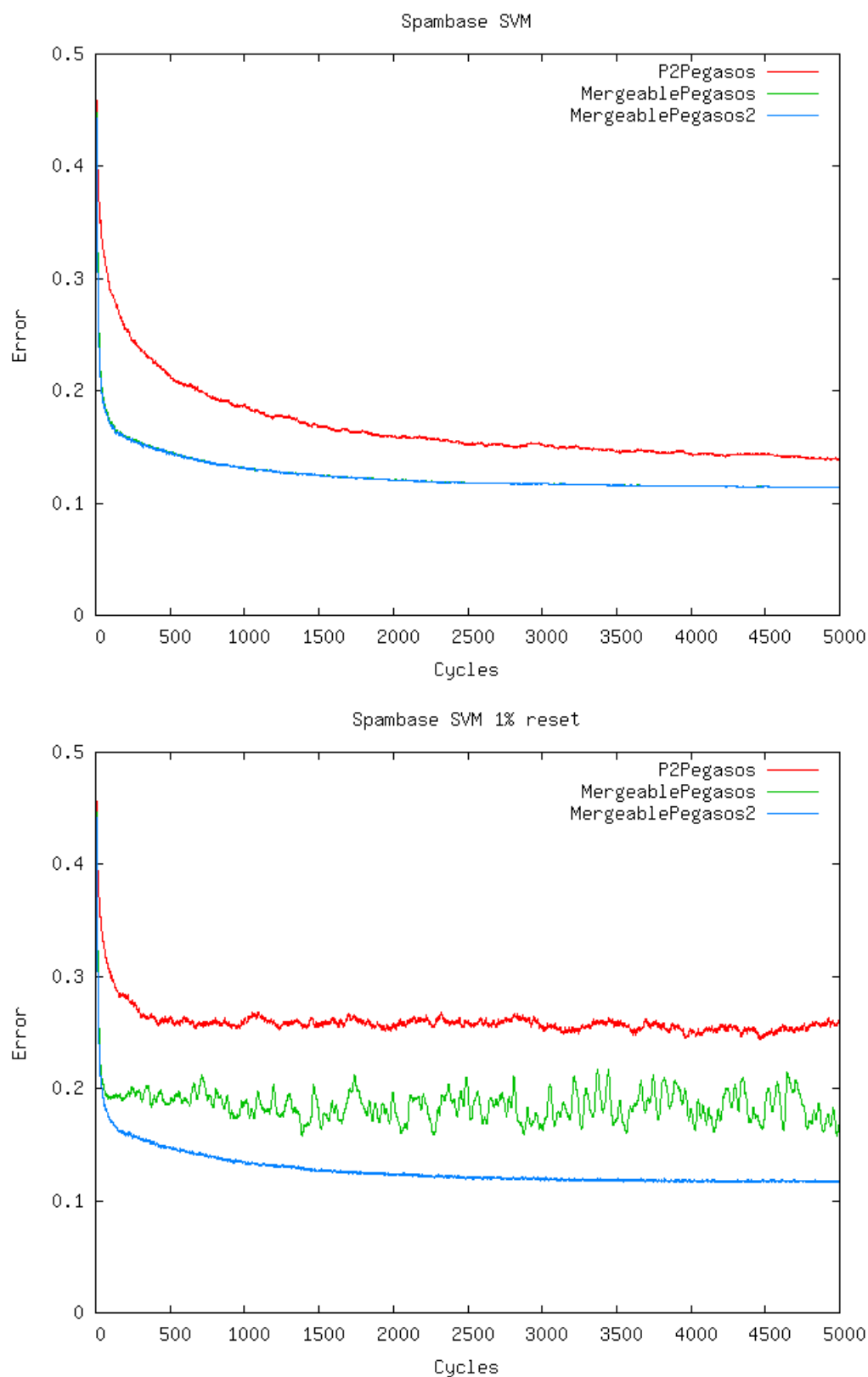
1%-os resetelődési valószínűséggel futtatva teszteket azt tapasztaltam, hogy ez jelentős teljesítményromlást okoz a vizsgált tanulók (*LogisticRegression*, *P2Pegasos*, *MergeableLogisticRegression*, *MergeablePegasos*) mindegyikénél. A nem összeolvasztható modellek esetén ennek az volt az oka, hogy a modellek átlagosan csak 100 iterációt éltek meg, utána mindig nulláról kellett előről kezdeniük a tanulást. Az összeolvasztható modellek esetén meg lett volna a lehetőség az információ átmentésére, hiszen ha a hálózaton egyes modellek el is felejtenek mindent, a többivel kombinálódva nagyjából visszanyerhetik azt, amit elvesztettek. (Összeolvasztható modellek alkalmazása esetén ugyanis meglehetősen homogénné válnak a hálózaton keringő modellek.) Ebben az esetben viszont az jelentette a problémát, hogy az összeolvastás során egyszerű átlag számítódik, azaz ugyanakkora súlyt kap a 2000 példát látott modell, mint az, amelyik csak egyet-kettőt látott. Így a hálózatba érkező iniciális modellek rendkívül lerontják a hálózaton lévő modellek teljesítményét.

Ezért létrehoztam a *MergeableLogisticRegression2* és *MergeablePegasos2* tanulókat, melyek abban különböznek a korábbiaktól, hogy a modellek összeolvastásakor súlyozott átlagot tekintek, és a súlyokat az életkorok adják. Így a nagyon fiatal modellek alig vannak hatással az idősebbekre, míg az idősebb modellek szinte felülírják a fiatalokat. Ezzel elenyészővé válik a resetelődés okozta teljesítményromlás, az információban keletkező "lyukak" hatékonyan betömődnek.

¹ A szimuláció nem tökéletes (pl. az overlay hálózatban már többen előre ismerni fogják az "új" eszköz címét), de a célnak megfelelő.



2.1. ábra. Nem összeolvasztható, összeolvasztható, valamint súlyozottan összeolvasztható logisztikus regresszió tanuló, spambase adatbázison tesztelve, 1%-os resettel (lent) illetve anélkül (fent). A felső ábrán a két összeolvasztható modell görbéje szinte egybeesik.



2.2. ábra. Nem összeolvasztható, összeolvasztható, valamint súlyozottan összeolvasztható SVM tanuló, spambase adatbázison tesztelve, 1%-os resettel (lent) illetve anélkül (fent). A felső ábrán a két összeolvasztható modell görbéje szinte egybeesik.

3. fejezet

Momentum

A szakirodalomban ismert technika a momentum, mely bizonyos esetekben képes felgyorsítani a sztochasztikus gradiens módszer konvergenciáját.[5] Lényege, hogy a korábban látott gradiens exponenciális mozgó átlagával frissítjük a modellt. A momentum implementálható az alábbi módon:

```
procedure update(s)
  p ← p + modell.gradient(s)
  modell.update(p * α)
  p ← p * (1 - α)
end procedure
```

Az s tanítópélda szerint kiszámítjuk a gradienst, azonban nem frissítjük ez alapján a modellt, hanem megnöveljük vele a p lendületet. Ezután a modellt frissítjük a lendület α részével (a modell aktuális életkorát figyelembe véve), és ugyanennyivel csökkentjük a lendületet. Az $0 < \alpha \leq 1$ paraméter határozza meg, mennyire gyorsan "felejt" az algoritmus. Az $\alpha = 1$ esetben a momentum nélküli változatot kapjuk vissza.

Felmerülhet bennünk a kérdés, hogy hosszan tartó tanítás során a fenti algoritmusban hány "gradiensnyi" lendület halmozódik fel (avagy mekkora a súlyok összege). Ezt MBS^1 -sel jelölöm, és a következő egyenlet megoldásával kapható meg:

$$MBS * (1 - \alpha) + 1 = MBS$$

Az egyenlet a lendülettel végzett műveleteket tükrözi, egyik *modell.update()* hívástól a másikig. Az egyenlet megoldása az alábbi:

$$MBS = 1/\alpha$$

Mivel a *modell.update()*-nek α -val megszorozva adjuk át a lendületet, láthatjuk, hogy az algoritmussal nem változott meg a lépésközök nagyságrendje.

A momentum módszert implementáltam a *MomentumLogisticRegression* és *MomentumPegasos* tanulókbán, azonban a teljesítménybeli változás mértéke elenyésző volt.

¹ "Maximal Batch Size", mivel a módszer tekinthető egyfajta fuzzy mini-batch-ként.

4. fejezet

Összeolvasható momentum

Minél nagyobb a hálózat, annál nagyobb a számítási kapacitásunk is. Ha viszont mindenki másmilyen modellt tanít, erőfeszítéseink szétforgácsolódnak. Összeolvasható modellek esetén szerencsére eléggé homogenizálódnak a modellek a hálózatban. Ha már ehhez a közös modellhez minden iterációban kiszámítunk rengeteg gradienst, érdemes volna ezeket valahogyan megosztani a peerek között. Ha mindenkire eljuttathatnánk mindenki más gradiensét, minden iterációban egy batch lépést hajthatnánk végre!

Az előbbi alapötlet egy közelítését adja az összeolvasható mini-batch algoritmus: Minden modell egy mini-batch-be gyűjtse a gradienseket; ha a gradiensek száma eléri az előre rögzített *MBS* korlátot, az összeg alapján hajtsuk végre az update lépést, és ürítsük a batch-et; összeolvashatóskor (a modellek átlagolása mellett) adjuk össze a batch-eket.

Ha feltesszük, hogy a batch méretek szinkronban maradnak¹, ez a módszer azt eredményezné, hogy kb. $\log_2 MBS$ iterációnként végrehajtunk egy kb. *MBS* méretű mini-batch lépést. Ha azonban csak a batch bő felét² "használnánk el" az update lépésre, (egy idő után) minden iterációban végrehajthatnánk egy kb. *MBS*/2 méretű mini-batch lépést.

Az összeolvasható mini-batch algoritmus egy elegánsabb alternatívája az összeolvasható momentum algoritmus. Minden lépésben használjuk el a batch bő felét a modell frissítésére. Csupán a batch összegét tároljuk: ez lesz a lendület. A pontos algoritmus az alábbi:

```
procedure merge(m)
  modell  $\leftarrow$  merge(modell, m.modell)
  p  $\leftarrow$  p + m.p
end procedure
```

```
procedure update(s)
  p  $\leftarrow$  p + modell.gradient(s)
  modell.update(p *  $\alpha$ )
  p  $\leftarrow$  p * (1 -  $\alpha$ )
end procedure
```

¹ Természetesen ez egy elég elrugaszkodott feltevés.

² Ezt érthetjük úgy is, hogy a gradiensek egy részhalmazát használjuk el, vagy úgy is, hogy mindegyik gradiensnek a felét.

Amikor a legfrissebben kapott modellbe akarjuk olvasztani az eggyel korábbi m modellt, a lendületeket adjuk össze, a modellek eredeti részén pedig végezzük el a szokásos összeolvasztást (ajánlatos a súlyozott változatot). A frissítés ugyanaz, mint korábban, de itt $\alpha > 1/2$.

Az MBS -re ezúttal az alábbi egyenlet teljesül:

$$MBS * (1 - \alpha) * 2 + 1 = MBS$$

Az egyenletet megoldva azt kapjuk, hogy:

$$MBS = 1/(2 * \alpha - 1)$$

Ez azt jelenti, hogy a lépésköz megnő az $\alpha/(2 * \alpha - 1)$ -szeresére. Azonban, ha valamilyen ok folytán nem minden alkalommal jön létre összeolvasztás, ennél kisebbeket lép, ami ésszerű is, hiszen kevesebb információ kevesebb magabiztosságot tesz lehetővé.

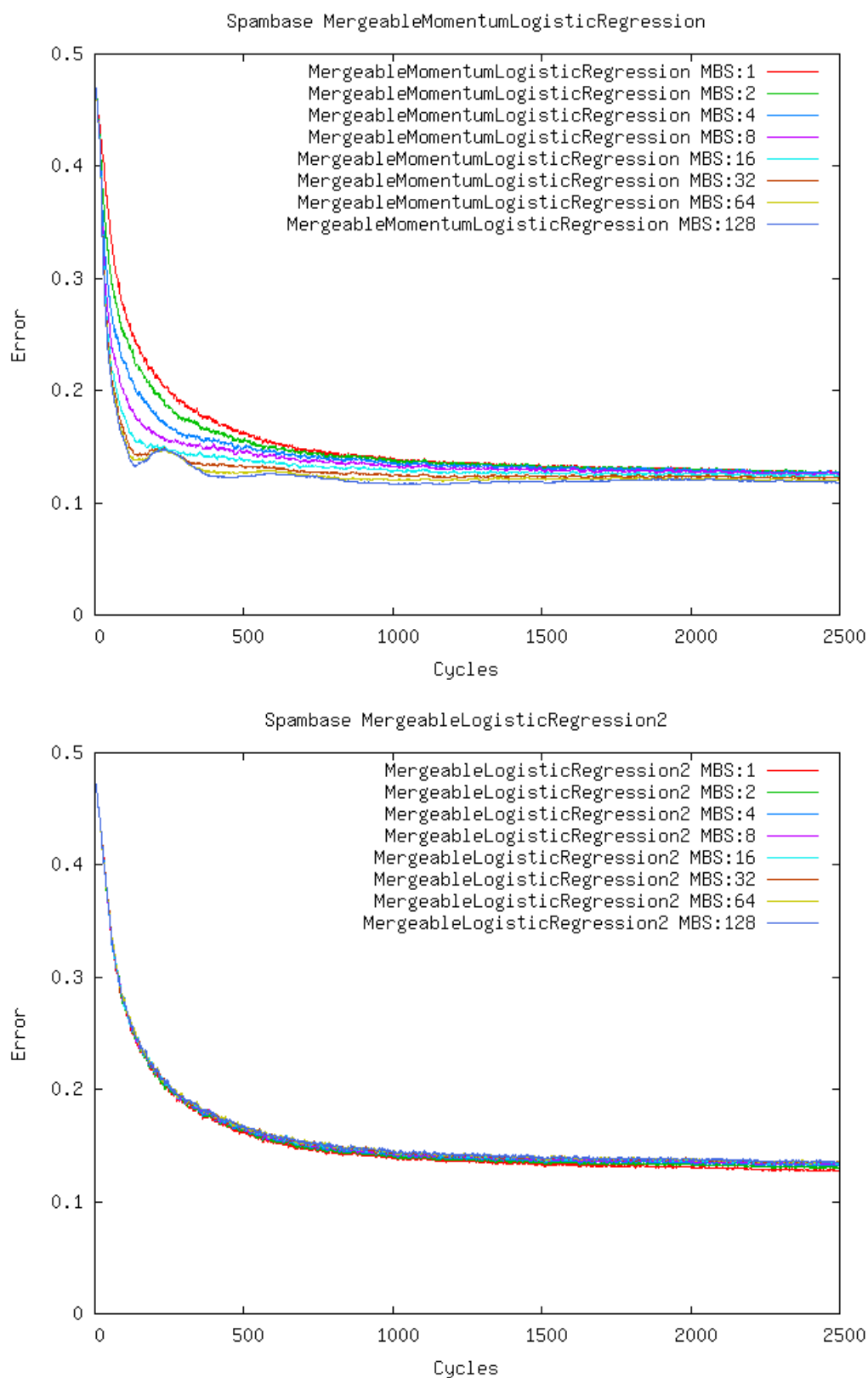
Az algoritmust megvalósítottam a *MergeableMomentumLogisticRegression* és *MergeableMomentumPegasos* tanulóknak. Célszerűbbnek láttam az MBS -t választani meta-paraméternek, és ebből számítani az α -t, az alábbi képlettel:

$$\alpha = (1 + 1/MBS)/2$$

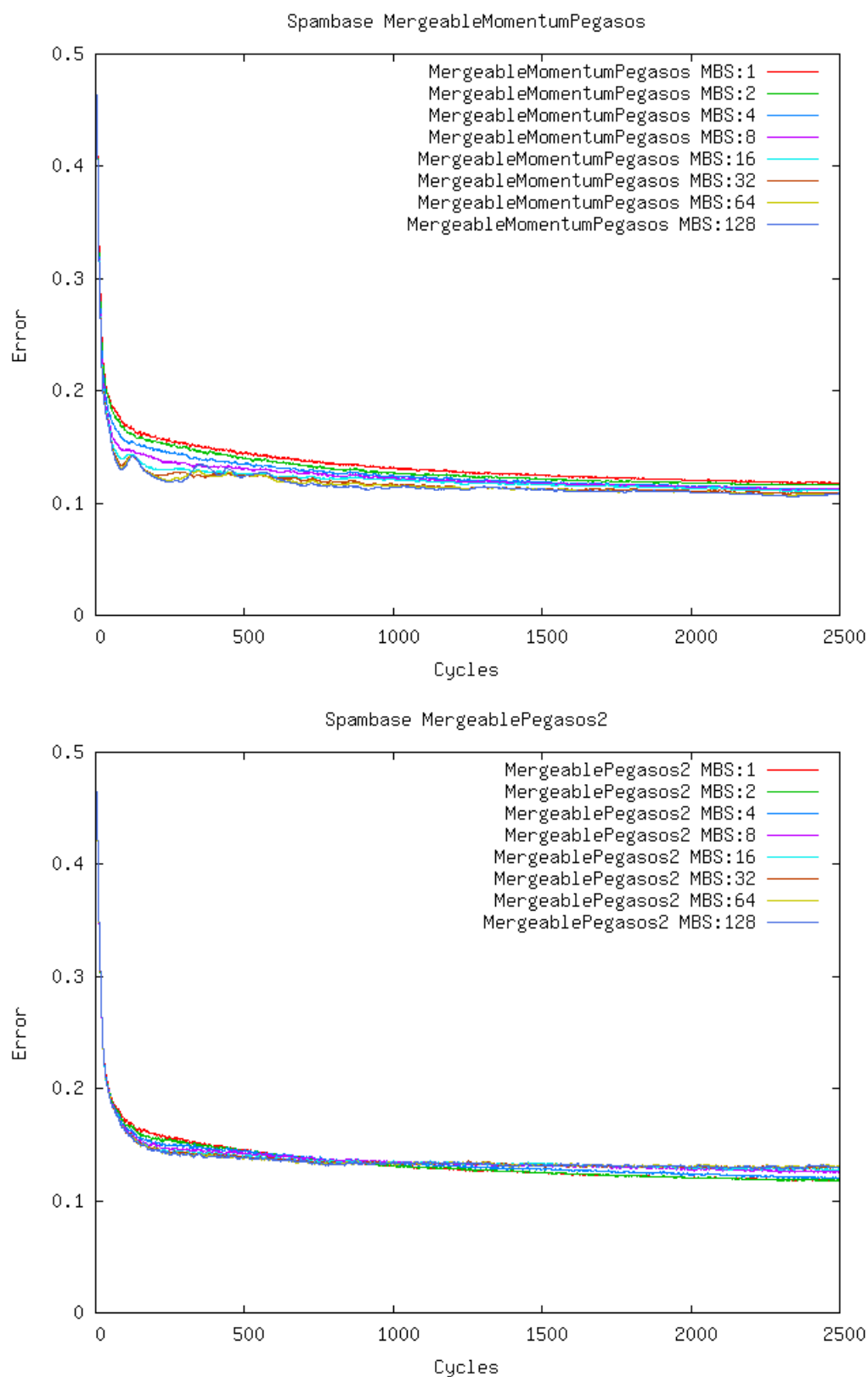
Az $MBS = 1$ esetben a momentum nélküli változatot kapjuk vissza.

Az összeolvasztható momentum számottevően javított a konvergencia sebességén. Annak érdekében, hogy megbizonyosodjak arról, hogy ez nem csak a megnövekedett lépésköznek köszönhető, kiértékeltem a *MergeableLogisticRegression2* és *MergeablePegasos2* tanulókat azon változatait is, amelyek lépésköze különböző MBS értékeknek megfelelő mértékben skálázott; de ez nem vált előnyükre.

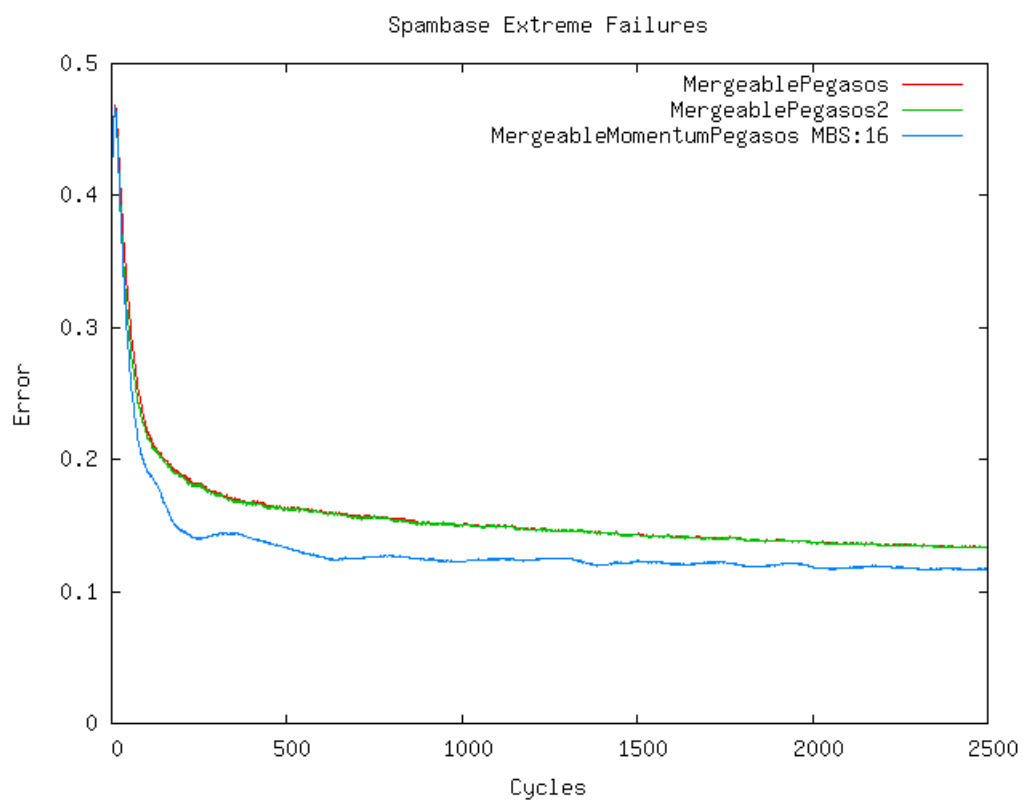
Az összeolvasztható momentum jó teljesítményt nyújtott extrém hibák (50%-os üzenetvesztés, 1-10 ciklusidőnyi továbbítási idő, 10%-os churn) mellett is.



4.1. ábra. Momentumos (fent) illetve momentum nélküli (lent) súlyozott összeolvasztható logisztikus regresszió tanuló, spambase adatbázison tesztelve, különböző *MBS* beállítá-
sokkal (momentum nélküli esetben ez csupán a lépésköz egy megfelelő skálázását jelen-
ti).



4.2. ábra. Momentumos (fent) illetve momentum nélküli (lent) súlyozott összeolvasztható SVM tanuló, spambase adatbázison tesztelve, különböző *MBS* beállítással (momentum nélküli esetben ez csupán a lépésköz egy megfelelő skálázását jelenti).



4.3. ábra. A különböző összeolvasztható SVM tanulók spambase adatbázison tesztelve, extrém hibákkal.

5. fejezet

Concept drift

Concept driftől[4] akkor beszélünk, ha idővel a megtanulandó minta változhat. Ez lehet egy lassú, fokozatos változás, vagy történhet hirtelen is. Ez azért jelent problémát, mert az előregedő modellek egyre kevésbé tudják követni a változásokat.

Erre egy lehetséges megoldás az AdaGoLF[4]: egy hagyományos tanulót átalakíthatunk úgy, hogy inicializáláskor generáljon egy véletlenszámot; amikor a modell életkora eléri ezt az értéket, a modellt újrainicializáljuk, és újabb véletlenszámot generálunk az eloszlásból (mely lehet pl. log-normális eloszlás). Ezzel a módszerrel a hálózaton keringő modellek életkora egy fix eloszlásra fog beállni, azaz biztosítható, hogy mindig legyen megfelelő életkorú modell, azaz olyan, mely kellően adaptív, ugyanakkor a konvergenciája is kedvező.

5.1. Concept drift a Gossip Learning Frameworkben

A GoLF képes szintetikus adatokkal folyamatos vagy hirtelen drift szimulációjára. A továbbiakban 100 peeres hálózatokon, 1000 iterációnként történő hirtelen drift mellett értékelem ki az algoritmusokat. A kiértékelés továbbra is egy elkülönített tesztalmazon történik, melyre ugyanúgy hat a drift.

A GoLF-ban megtalálható az AdaGoLF implementációja, a *SelfAdaptiveModel*, mely wrapperként használható más tanulókkal.

6. fejezet

Összeolvasható AdaGoLF

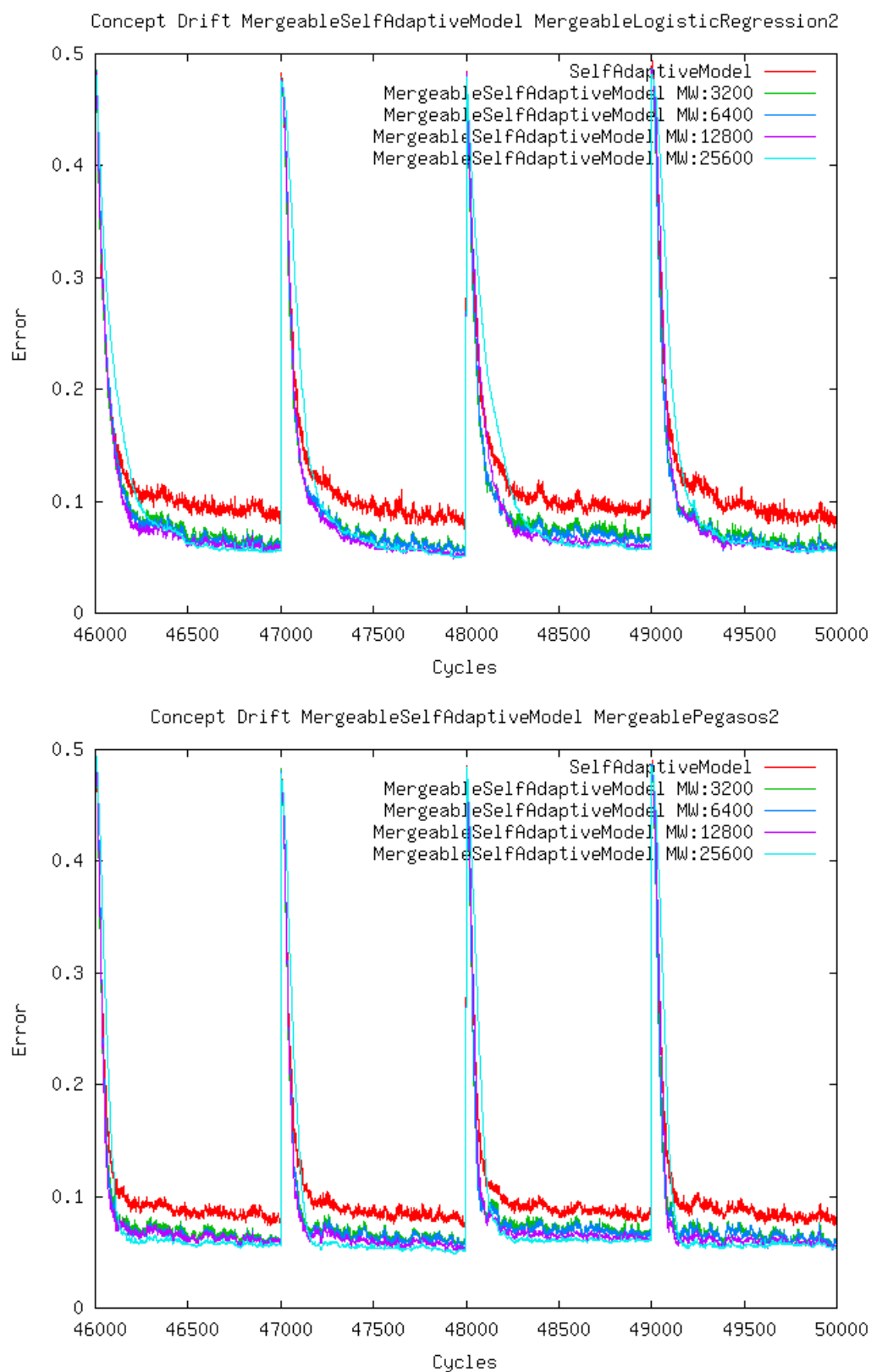
Korábban láthattuk, hogy a momentum módszer csak összeolvasztással együtt volt hatékony. Így ahhoz, hogy concept driftes környezetben is hasznosítható legyen, először magát az összeolvasztást kell megoldani. Ez azért problémás, mert az összeolvasztás homogenizálja a modelleket, míg az AdaGoLF a diverzitásra épül.

Erre egy kompromisszumos megoldást, a *MergeableSelfAdaptiveModel*-t javaslom: Az időt bontsuk fel egyenlő, *MergeWindow* hosszú intervallumokra, és alkalmazzunk súlyozott összeolvasztást azon modellek esetén, melyek inicializálási ideje azonos intervallumba esik.

Ezáltal létrejönnek határvonalak, melyeken túl az idős modellek már nem tudják befolyásolni a fiatalabbakat, ugyanakkor korlátozott mértékben kihasználhatóvá válnak az összeolvasztás előnyei. Lényeges, hogy nem egyszerűen csak az inicializálási idők különbségét hasonlítjuk össze egy küszöbértékkel, mert úgy áttételesen korlátlanul tudnának terjeszkedni az idős modellek.

A *MergeWindow* ideális beállítása függ a drift sebességétől. Ha túl kicsire állítjuk, kevés lehetőség lesz összeolvasztásra, így lassabban fog konvergálni. (Hasonlítani fog a *SelfAdaptiveModel*-re.) Ha túl nagyra állítjuk, nem lesz eléggé adaptív, azaz a drift megtörténte után lassan kezd javulni a teljesítménye. (Hasonlítani fog a hagyományos tanulóakra.)

A tesztek tanulsága szerint ezzel a módszerrel a *SelfAdaptiveModel*-énél jobb teljesítmény érhető el.



6.1. ábra. Súlyozottan összeolvasztható logisztikus regresszió (fent) illetve SVM (lent) tanulók *SelfAdaptiveModel*be, valamint *MergeableSelfAdaptiveModel*be ágyazva, utóbbi esetben különböző *MergeWindow* beállításokkal.

7. fejezet

Modellcsomag módszer

A pletyka tanulást mobil eszközökön alkalmazva az tapasztalható, hogy elsősorban az akkumulátorhasználat szab korlátokat. Ezen belül is az adóvevő üzenetküldéshez szükséges magas teljesítményre kapcsolása a legjelentősebb tétel, nem is maga a kommunikáció (amíg kevés adatot küldünk).

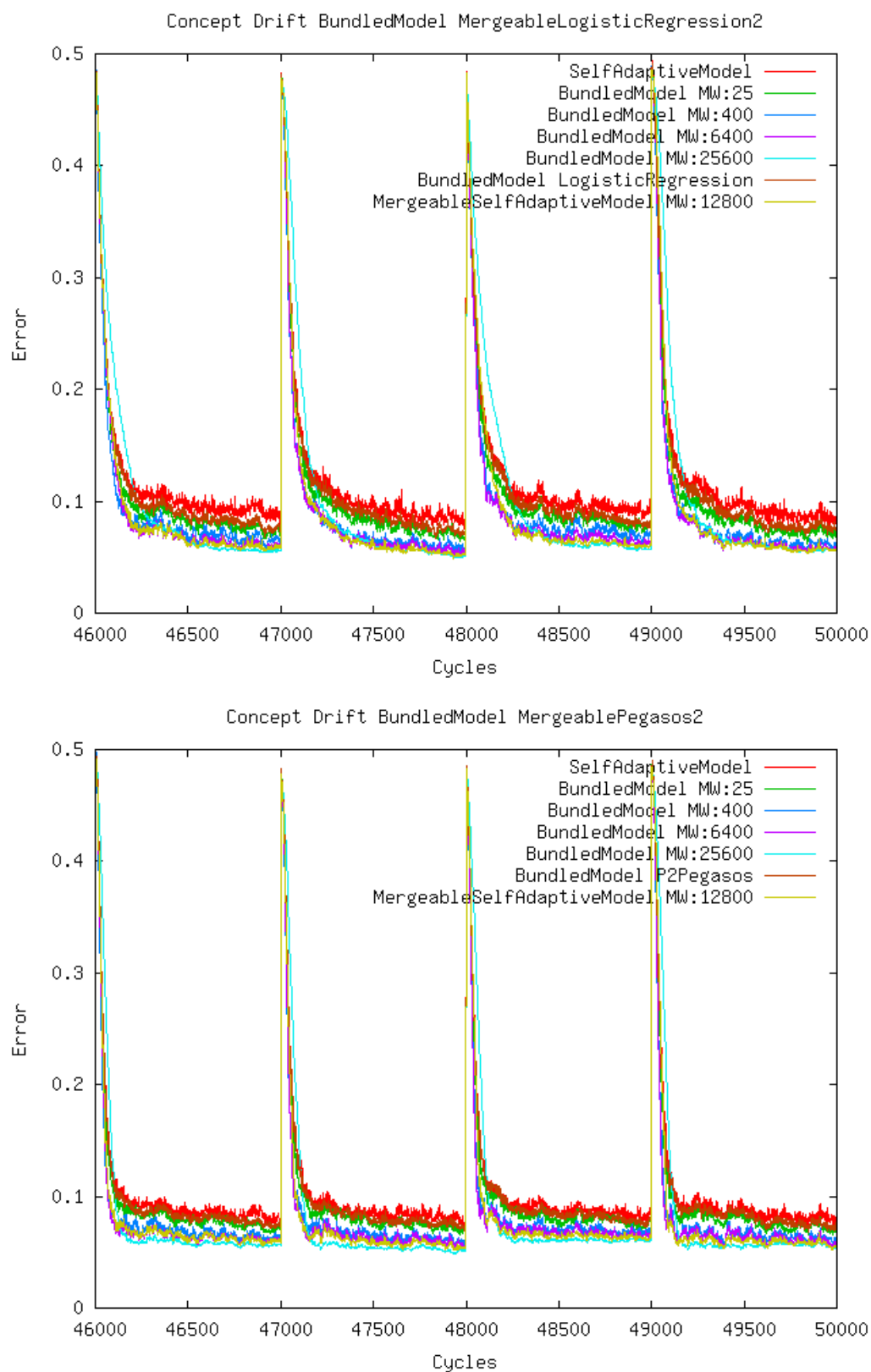
Ebből az következik, hogy nem jár jelentős költségnövekedéssel, ha egyszerre több modellt küldünk. Az ilyen modellcsomagokat alkalmazhatjuk több *MergeableSelfAdaptiveModel* együttes továbbítására, amivel növelhetjük az egy modellre eső összeolvasztások várható értékét.

Így kidolgoztam a *BundledModelt*, mely *MergeableSelfAdaptiveModel*ek wrapperjeként funkcionál, a következők szerint: tanításkor mindegyik modellt tanítja; predikcióhoz a legjobbnak vélt modellt¹ használja; összeolvasztáskor véletlenszerűen kialakítja a lehető legtöbb párt, és ezeken elvégzi az összeolvasztást.

A tesztjeim során 10-es csomagokkal dolgoztam, és kipróbáltam többféle *MergeWindowt*. Csupán kis mértékben sikerült javítani az önmagában alkalmazott *MergeableSelfAdaptiveModel*hez képest, ami arra utal, hogy már eleve aránylag sok összeolvasztás történik. SVM esetén mégkevésbé látszódnak a különbségek.

Annak kiderítése érdekében, hogy mekkora előnyt jelent pusztán az, hogy több modellből választjuk ki a legjobbnak tűnőt, végeztem olyan tesztet is, ahol *SelfAdaptiveModel*-eket raktam *BundledModel*be. Ez kis mértékű javulást eredményezett az önmagában alkalmazott *SelfAdaptiveModel*hez képest.

¹ Ennek megállapításához lokálisan tesztelhetjük a modelleket a tanítópéldákon, mielőtt még tanítotuk volna velük a modelleket.[4]



7.1. ábra. Súlyozottan összeolvasztható logisztikus regresszió (fent) illetve SVM (lent) tanulók *BundledModel*be foglalt *MergeableSelfAdaptiveModel*ekbe ágyazva, különböző *MergeWindow* beállításokkal, továbbá néhány baseline.

8. fejezet

Forgó csomag módszer

Hogy maximalizáljam az összeolvasztások számát, kidolgoztam egy más elven működő csomagot is, az *RotatingBundlet*. Ez már önmagában biztosítja az adaptivitást, tehát *MergeableSelfAdaptiveModel* helyett közvetlenül¹ a konkrét tanulót rakjuk bele. A modellben lévő helyek sorszámozottak, és előre meghatározott δ időközönként rotálunk bizonyos modelleket, továbbá van egy globális sorszámozottsága a rotációknak is, és eltávolítjuk az utoljára elvégzett rotáció sorszámozását.

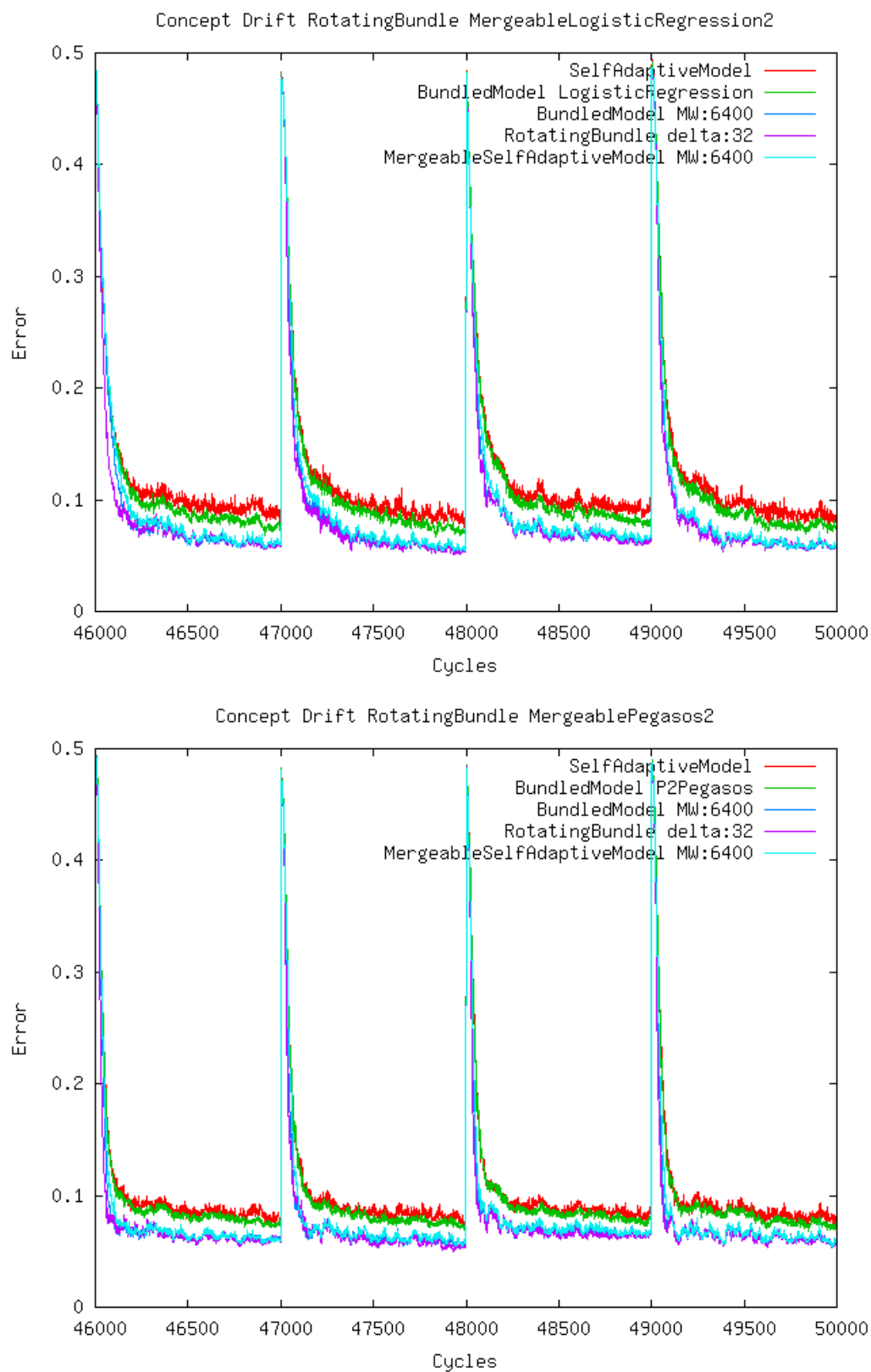
Egészen pontosan az történik, hogy megnézzük, a rotáció sorszáma binárisan felírva hány nullára végződik; az első ennyi modellt eltoljuk eggyel nagyobb pozícióra, és amelyik modell így felülíródna, azt újrainicializálva visszahelyezzük az első helyre. (Egyesre végződő bináris alak esetén egyszerűen az első helyen lévő modell inicializáljuk újra.)

A rotációknak igazából nem kell szinkronban történniük, végezhetjük lusta módon: update előtt az aktuális időnek megfelelő állapotba hozzuk a modellt a megfelelő számú rotációt elvégezve, valamint merge előtt a lemaradt modellt beforgatjuk a másikhoz igazodva. Így nem fontos az sem, hogy nagyon pontosak legyenek a peerek órái. A tanítás és a predikció a *BundledModel*hez hasonlóan történik, összeolvasztáskor pedig az egymásnak megfelelő pozícióban lévő modelleket olvasztjuk össze.

A modellek az egyes pozíciókon így kettőhatvány időket töltenek el, és a koruk is közel kettőhatvány lesz. Ezáltal 10 model esetén akár napostól éves szintű drift is kezelhető.

A tesztelés során a paraméterre nézve nagy robosztusságot mutatott. De ismét csak igen kis mértékben sikerült javulást elérni az előző megoldáshoz (*BundledModel*) képest.

¹ Egy teljesítménymérést végző *Wrapper* osztályra még szükség van.

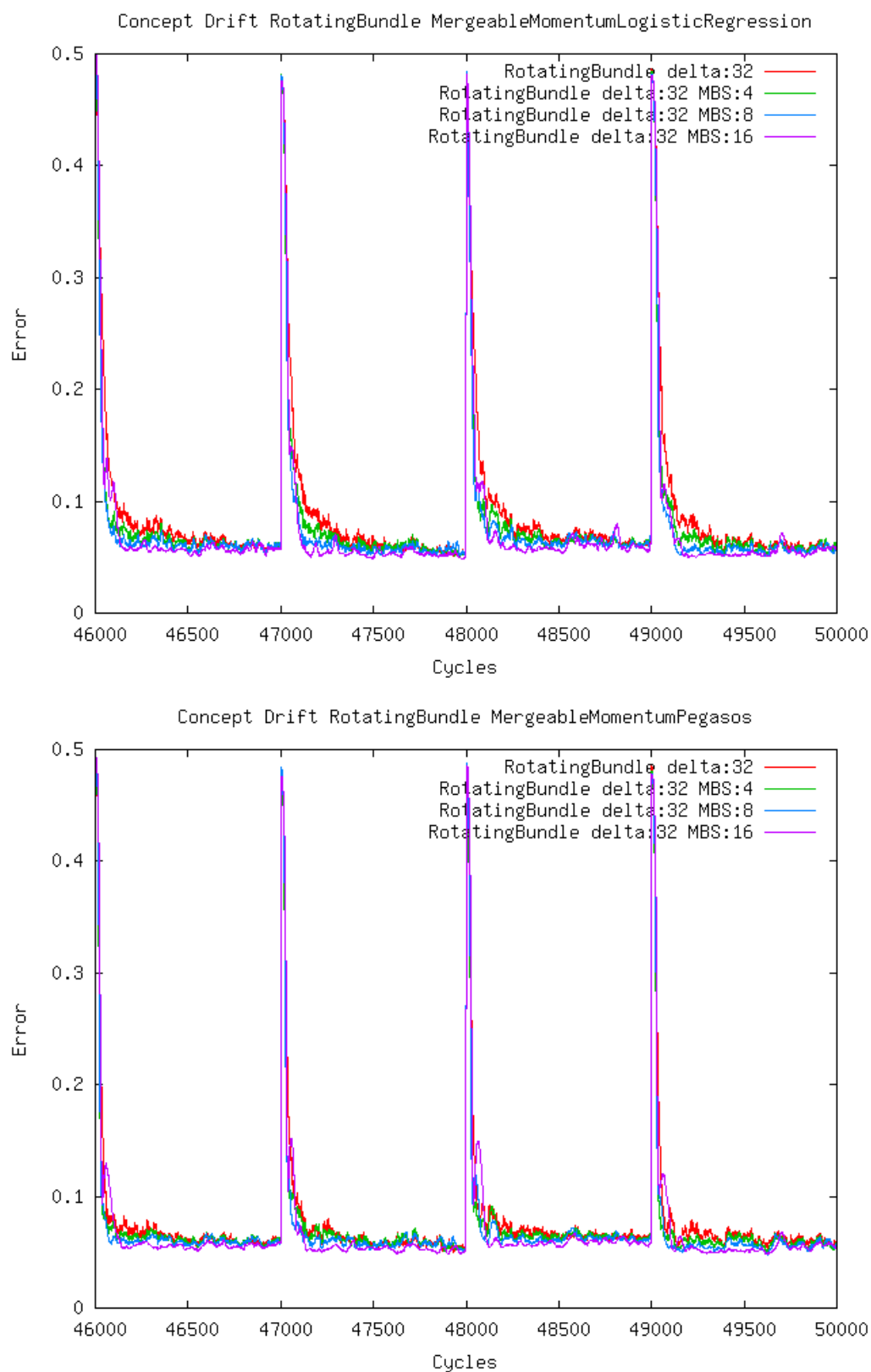


8.1. ábra. Súlyozottan összeolvasztható logisztikus regresszió (fent) illetve SVM (lent) tanulók *RotatingBundle*-be foglalva, összehasonlítva korábban tárgyalt megoldásokkal.

9. fejezet

Momentum és concept drift

Végül a *RotatingBundle* és az összeolvasztható momentum kombinálásával megkapjuk az eddigi legjobb concept driftes eredményeket.



9.1. ábra. Összeolvasztható momentumos logisztikus regresszió (fent) illetve SVM (lent) tanulókat *RotatingBundle*-be foglalva, különböző *MBS* értékekkel, valamint összehasonlításként a momentum nélküli változat.

Nyilatkozat

Alulírott programtervező informatikus MSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Tanszékcsoport Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszékén készítettem, programtervező informatikus MSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy diplomamunkámat a Szegedi Tudományegyetem Informatikai Tanszékcsoport könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2014. május 17.

.....
aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani Jelasity Márknak az érdekes témáért, tanácsaiért, türelméért; Hegedűs Istvánnak és Gub Jánosnak a technikai segítségükért; valamint Ormándi Róbertnek, hogy invitált a projektbe.

Irodalomjegyzék

- [1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer Science+Business Media, LLC, New York, 2006.
- [2] N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, Cambridge, 2000.
- [3] R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification*, Wiley-Interscience, 2000.
- [4] I. Hegedűs, R. Ormándi, M. Jelasity, Massively Distributed Concept Drift Handling in Large Networks. *Advances in Complex Systems*, **16** (2013).
- [5] N. Le Roux, M. Schmidt, F. Bach, A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. *NIPS*, 2012.
- [6] R. Ormándi, *Applications of Support Vector-Based Learning*, PhD thesis, Szeged, 2013.
- [7] R. Ormándi, I. Hegedűs, M. Jelasity, Asynchronous Peer-to-Peer Data Mining with Stochastic Gradient Descent. *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011), Lecture Notes in Computer Science*, **6852** (2011), 528–540.
- [8] Ormándi, R., Hegedűs, I., Jelasity, M., Gossip learning with linear models on fully distributed data. *Concurrency Computat.: Pract. Exper.*, **25** (2013), 556–571.
- [9] S. Russell, P. Norvig, *Artificial Intelligence. A Modern Approach*, Pearson Education, Inc., New Jersey, 2003.