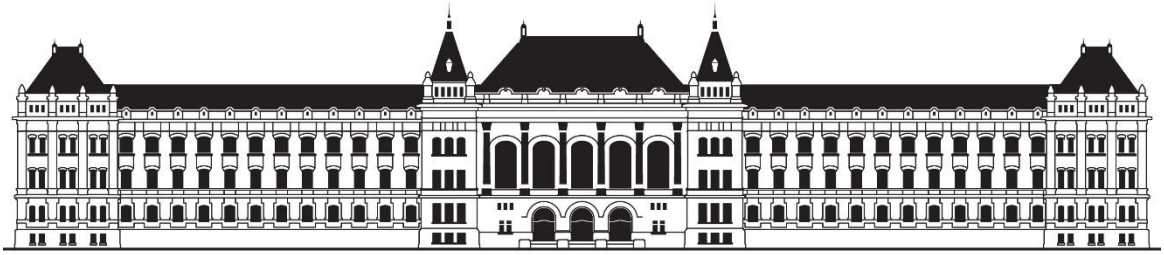


MŰEGYETEM 1782

ÖNÁLLÓ LABORATÓRIUM

Adatvédelmet biztosító federált tanulás súlyozott kliensekkel

Készítette: Jakab Máté

Konzulensek: Ács Gergely és Horváth Máté

TARTALOMJEGYZÉK

1. MOTIVÁCIÓ, PROBLÉMA MEGHATÁROZÁSA

Az Önálló laboratórium tárgy keretén belül a gépi tanulás egy adott területével ismerkedtem meg, melynek segítségével elérhető, hogy ugyanazt a feladatot neurális háló használatával megoldani kívánó kliensek (pl. cégek) egymást segítsék anélkül, hogy a bizalmas adataikat egymással (vagy bárki mással) megosszák. Ezzel a módszerrel egy adott feladatra specifikált modellt közösen tudnak fejleszteni erőforráshatékonyan (egy szerver segítségével).

A félévi munkám fontos szempontja volt az, hogy a közös munkában résztvevők a hatékonyságuk alapján súlyozva legyenek, és a súlyozás alapján legyen számítva a közös modell. A hivatkozott dokumentumban [1] is felmerül a súlyozás kérdése, ahol a súlyozás nem a hatékonyság, hanem az egyes kliensek számára elérhető helyi adataik mennyisége alapján történik. Emellett a felmerülő adatvédelmi problémákra is megoldásokat kerestem.

Adott cégek (pl. gyógyszergyártó cégek) rendelkeznek adatbázisokkal, és ezeket az adatbázisokat felhasználva be tudnak tanítani egy modellt (pl. neurális hálót), amely a múlt adatait felhasználva egy adott kérdésben egy jövőbeli dologról dönteni tud (pl. adott paraméterek alapján meg fog-e halni egy adott ember 24 órán belül vagy sem). Viszont nem minden esetben rendelkeznek a cégek elegendő méretű/hasznosságú adatbázissal, és ekkor felmerül az igény arra, hogy más cégek adatbázisait is felhasználják.

Könnyen belátható, hogy ez előrelépést jelentene, ugyanis több múltbeli adat nagyobb rálátást adna egy adott kérdésben, pontosabb jóslásokat kaphatnának. Viszont a cégek által – akár évek alatt összegyűjtött – adat rendkívül nagy értékű, nem lenne észszerű, ha ezeket az adatokat kiadnák, továbbá adatvédelmi szempontokat is sértene ezen adatok publikálása. Továbbá az adatok nagy mérete és az erőforrások limitjei miatt az a kivitelezhető, ha a cégek kizárólag a saját adataikkal számolnak. A módszer, amivel a fent említett feltételek nem sérülnek, az a *federált tanulás*.

Federált tanulás esetén adott egy központi *szerver*, amely a cégek (innentől *kliensek*) számára elküld egy modellt, amelyet minden kliens (vagy az összes kliensből n darab) az adott tanító adathalmazuk segítségével frissít.

Ezeket a frissült modelleket a kliensek visszaküldik titkosítva a szerver számára, amely kizárólag az aggregált/központi modellt tudja visszafejteni, a kliensek modelljeit külön-külön nem. Ennek a protokollnak *biztonságos összesítés* (vagy angolul *secure aggregation*) a neve.

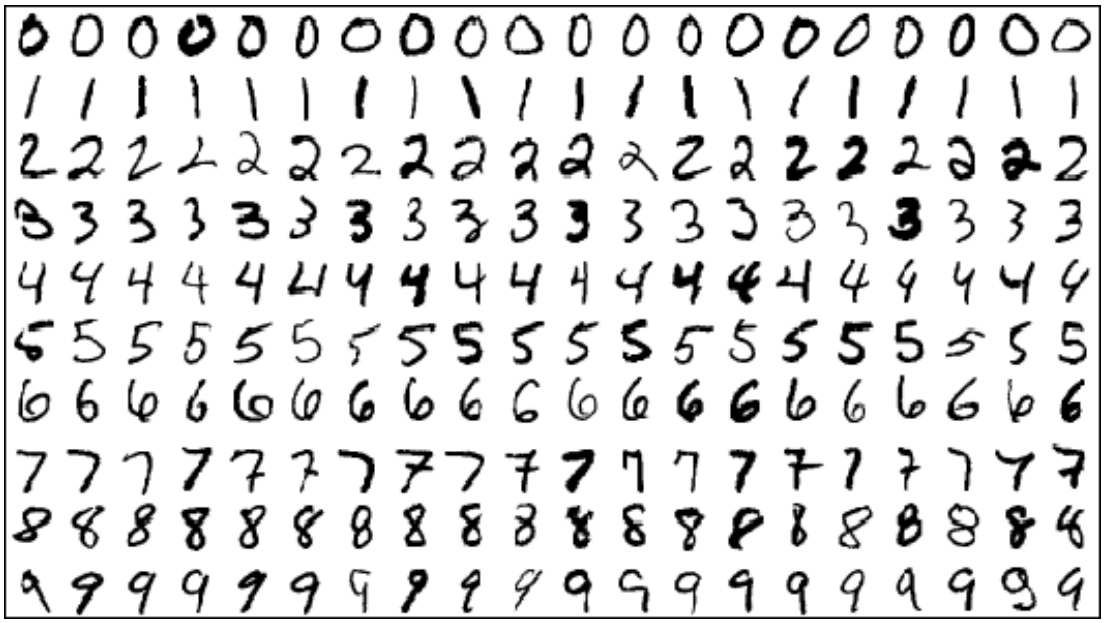
Látható, hogy ebben az esetben a kliensek ugyanolyan arányban vesznek részt a központi modell frissítésében. Az ennek következtében történő torzítás (amely lehet szándékos és nem szándékos is) kiküszöbölése érdekében a szervernek valamilyen teljesítménymutató alapján súlyoznia kell a klienseket, és ezek alapján kell kiszámolnia, hogy mi legyen a központi modell.

A fentiek alapján két főbb probléma megoldására törekedtem. Az egyik kliensek súlyozása volt, ahol a fő kérdés az volt, hogy milyen algoritmus alapján döntsük el, hogy melyik kliens kaphat nagyobb súlyt a központi modell számításakor. A másik az adatvédelem biztosítása volt, amelyet többek között a *funkcionális titkosítás* segítségével lehet kivitelezni.

2. SZÜKSÉGES HÁTTÉRTUDÁS

Ahhoz, hogy érdemben foglalkozni tudjak a témával, meg kellett ismerkednem a mesterséges intelligencia és a gépi tanulás alapfogalmaival (pl. input, output, final hypothesis, neural network, feature, epoch, batch, prediction stb.). Ezeket nem fejtem ki részletesen a dokumentum során.

A korábban említett cégek bizalmas adatbázisát egy publikusan elérhető csomaggal helyettesítettem a munkám során (erről lesz részletesebben szó, hogy pontosan hogyan), melynek a neve *MNIST* (Modified National Institute of Standards and Technology). Ez az adatbázis 28x28-as képfájlokat tartalmaz kézzel leírt számjegyekről, ezeket tekintjük a változóknak és ezekhez tartozik egy-egy érték (0, 1, … 9). Az MNIST a gépi tanulás „Hello world!”-je.



1. az MNIST adatbázisában található képek kézzel írt számjegyekről
Forrás: [8]

A neurális hálók legegyszerűbben a pythonban megírt *Keras* könyvtár segítségével implementálhatók, ezt használtam én is. A könyvtár függvényeiről részletesen nem lesz szó.

Az AdaBoost algoritmus egy népszerű algoritmus a mesterséges intelligenciában.

Az ElGamal titkosítás egy Diffie-Hellman kulcscserén alapuló aszimmetrikus titkosító algoritmus a nyilvános kulcsú kriptográfiában.

3. MEGOLDÁS

3.1 Federált tanulás adatvédelmi szempontok nélkül

Federált tanulás során különítsük el a szervert és a klienst. A szerver feladata az, hogy eljuttasson egy kezdetben random paraméterekkel inicializált modellt a kliensek számára. A kliensek a saját adatbázisuk segítségével tanítják, majd visszaküldik a szervernek a frissített modellt. A szerver megkapja ezeket a frissített modelleket, aggregálja őket. Röviden összefoglalva ez a folyamat zajlik le federált tanulás esetén egy kör alatt.

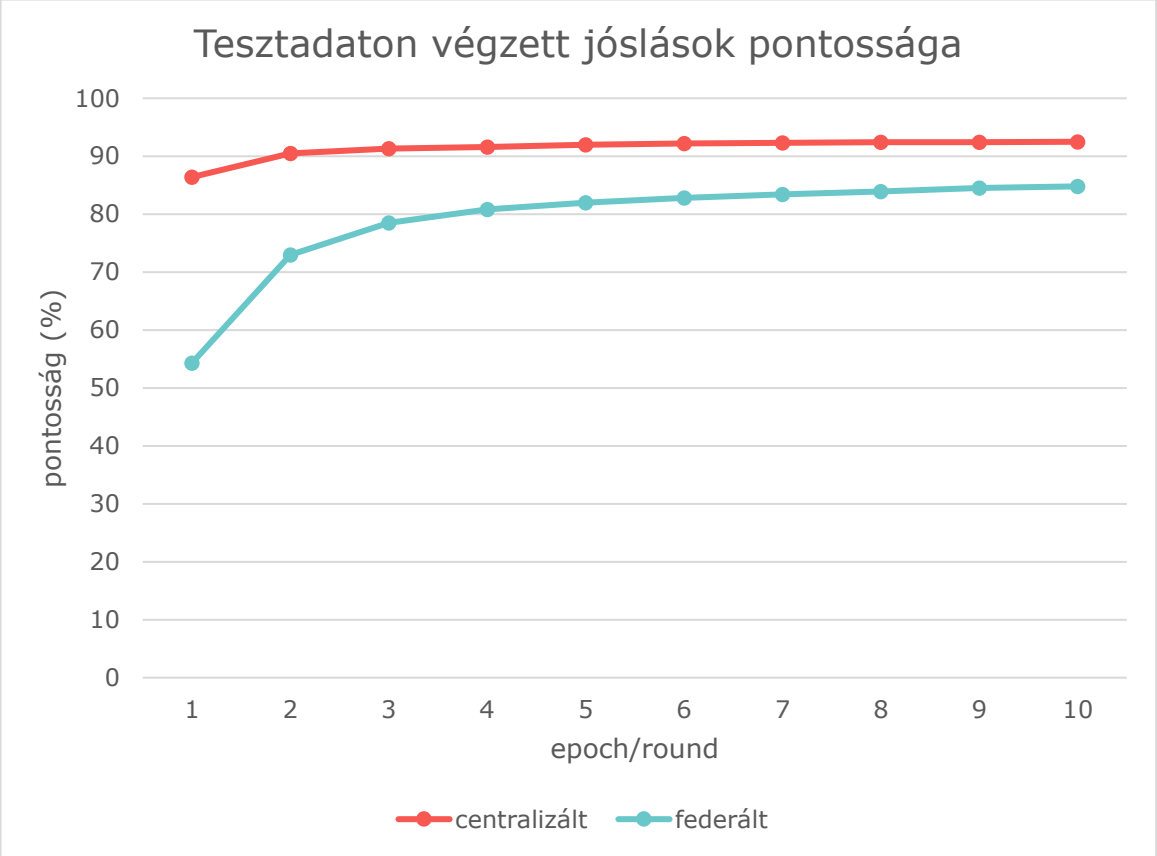
Amit érdemes megemlíteni, hogy amennyiben túl nagy a kliensek száma, akkor optimalizáltabb megoldást jelenthet az, ha egy körben nem az összes kliens vesz részt a tanításban, csak néhány százaléka.

A korábban említett MNIST adatbázis feldarabolásával hoztam létre több részadatbázist (külön-külön fájlokba mentve), melyekhez egy-egy klienst dedikáltam. A számjegyfelismerés szempontjából kb. azonos értékű adatbázissal rendelkező kliensek jöttek így létre.

Kezdetben a kliensek súlyozása nem volt szempont, csak a federált tanulás hatékonyságáról akartam meggyőződni, ezért a szerver által történő aggregálás során minden kliens ugyanakkora súllyal vett részt az összesített modell kiszámolásakor.

Megvizsgáltam, hogy hogyan teljesít a federált tanulás a centralizált tanulással szemben. Azért, hogy egyenlők legyenek az esélyek, a federált tanulásnál minden körben kiválasztásra került az összes kliens, a klienseknél az epochok száma 1 volt. Így a jelen példában federált tanulás esetén egy körben 100x600 tanuló adat állt rendelkezésre, míg centralizált esetben 1x60000. Ebben az esetben a federált tanulás egy köre megfeleltethető a centralizált tanulás egy epoch-ának.

Az eredményeket összehasonlítottam az MNIST 10000 mintából álló tesztadatbázisán végzett jóslások pontossága szerint. A következő grafikonon ábrázoltam az eredményt egy lefutásra. A vízszintes tengely a federált tanulás esetében az egyes körök (round) eredményeit jelzik, a centralizált tanulás esetében pedig az epochok számát. A federált tanulásnál a



A grafikonról leolvasható, hogy amikor az összes adattal dolgozunk (centralizált adatbázis), akkor már 2 epoch után eléri a maximumát (90% körüli pontosság), a federált tanulás esetén ezt az értéket 10 round után sem éri el, de egészen megközelíti (~85%). Elmondható, hogy lassabban éri el a maximum pontosságot, mint centralizált tanulás. Természetesen ez annak köszönhető, hogy a kliensek kisebb adatbázissal tanítanak, méghozzá minden körben egymástól függetlenül.

Tehát elmondható, hogy van jobb megoldás, de a federált tanulás egészen jól teljesít. Emiatt a fentebb már leírt problémánk megoldására kiválóan alkalmazható.

A továbbiakban az egy körben kiválasztott kliensek számát csökkentettem (25%-ra) a gyorsabb tesztelések érdekében.

3.2 Kliensek súlyozása

A szervernek a feladata, hogy valamilyen módon súlyozza a klienseket, hogy ne ugyanakkora súllyal vegyenek részt a közös modell frissítésében az egyes kliensek. Ennek az az oka, hogy ki kell szűrnünk az esetleges támadásokat is. El kell kerülnünk azt, hogy ugyanakkora „beleszólása" legyen egy olyan kliensnek, aki nagyságrendekkel kisebb tanuló adatbázissal rendelkezik, vagy ki kell szűrnünk azokat a klienseket, akik hibás adatokkal tanítanák a modellt (rosszindulatú támadás). Lehetnek a kliensek között ún. freeriderek/ghostok, akik nem járulnak hozzá a közös munkához, nem tanítják a körök során a kapott modellt, ezeket is el kell különítenünk a „hasznos", jól tanító kliensektől egy megfelelő súlyozással.

Ehhez a szervernek szükséges tudnia azt, hogy hogyan teljesítettek a kliensek egy-egy körben. Ezt természetesen nem „kérdezheti meg" a kliensektől, erről a szervernek kell döntenie, a kérdésnek a kliensektől függetlennek kell lennie. A súlyozásnál használt a teljesítménymutatónak a *pontosságot* választottam. Jelen esetben a pontosság alatt azt értem, hogy a szerver tesztadatbázisában lévő képek hány százalékánál találta el egy-egy kliens azt, hogy melyik érték tartozik hozzá. A pontosság természetesen 0 és 1 közé fog esni.

$$0 \leq a\ kliens\ pontossága\ = \frac{helyesen\ értékkel\ jósolt\ képek\ száma}{összes\ tesztadat} \leq 1$$

A megoldásom során többféle algoritmust is teszteltem, melyekben minden esetben szerepelt a fent említett pontosság mint változó. A kliensek által frissített modellek súlyainak az összege minden esetben 1. Ehhez a súlyozás során mindig normalizálni kell az algoritmusok végén a kapott súlyokat, hogy összegre vonatkozó feltétel teljesüljön.

Megfontolandó döntés volt, hogy a kliensek múltja befolyásolja-e a súlyozó algoritmust, tehát ha pl. az előző körben rosszul teljesített, akkor feltételezzük-e, hogy a következő körben is rosszul fog teljesíteni. A kérdésünk a következő: valószínűsíthető-e, hogy a körök között módosul egy kliens tanuló adatbázisa, mellyel javulni a pontosság?

Mivel a megoldásomban minden kliens a tanulás kezdete előtt megkapja a fix adatbázisát, ezért érdemes figyelembe venni a múltat, azaz az előző körben kapott súlynak szerepelnie kell a súly kiszámításának képletében. Ez azzal magyarázható, hogy nem fognak körönként változni (pl. nem lesz támadóból

jól teljesítő kliens a futás során). Természetesen az első körben ez az érték mindenkinél 1/k, ahol k az egy körben kiválasztott kliensek száma.

Az általam kiválasztott algoritmusok, amelyeket vizsgálni fogok a következők:

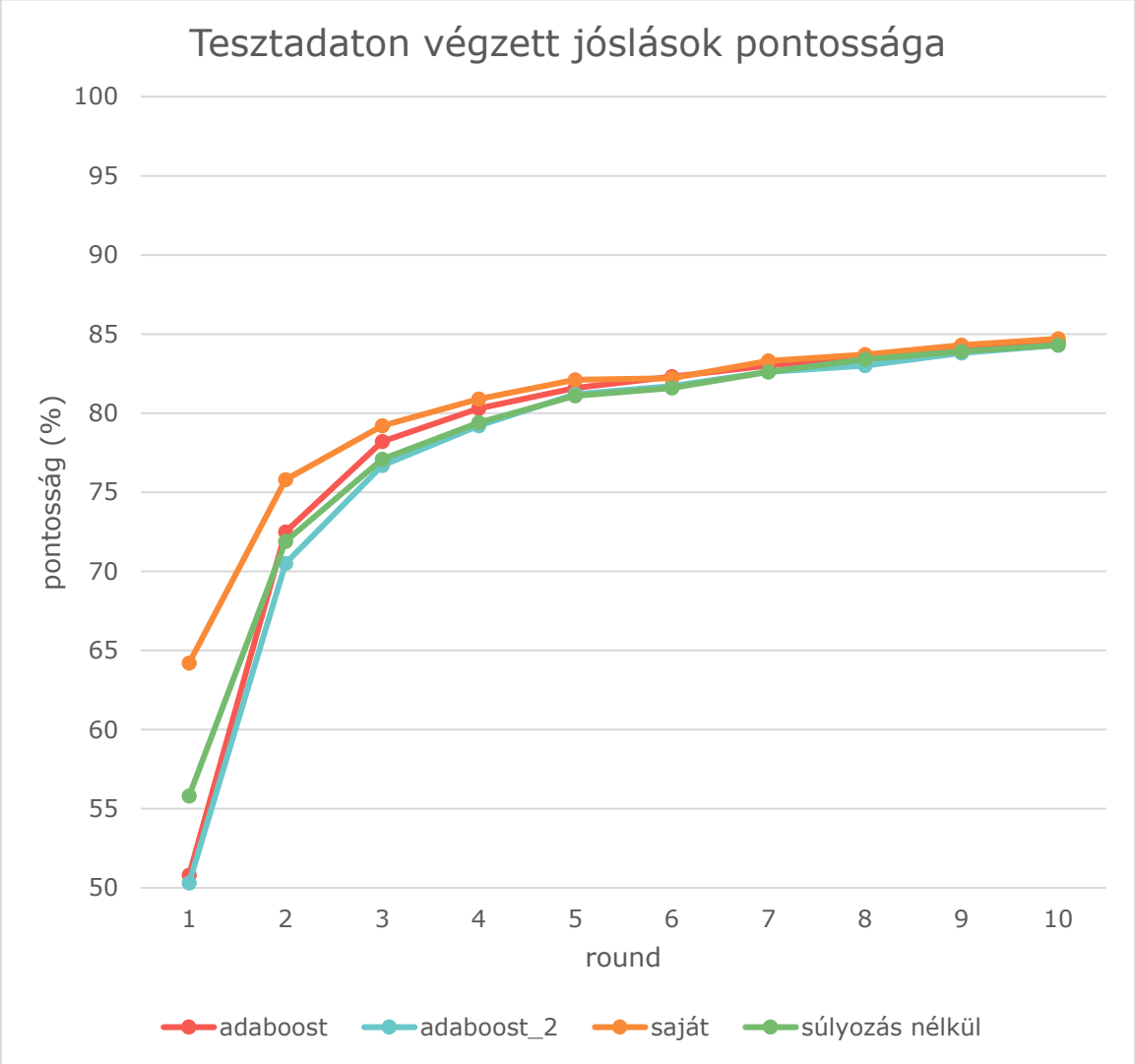
- AdaBoost technológiában használt képletek felhasználása – (adaboost)
    - kiszámoltam minden kliens „beleszólásának” mértékét (amount of say) az alábbi képlet szerint:
        - $amount\_of\_say = 0{,}5 * ln\frac{accuracy}{1-accuracy}$
    - ezzel kaptam egy értéket, amely *e* kitevőjében jelent meg. ezt megszoroztam az előző súllyal:
        - $weight = previous\_weight * e^{amount\_of\_say}$
    - azt értékeket normalizáltam, hogy az összegük 1 legyen
- AdaBoost technológia átépítése a federált tanulásba: a fent kiszámolt normalizált értékeket valószínűségnek tekintve véletlenszerűen kiválasztunk k darabot, ahol k az egy körben kiválasztott kliensek száma – (adaboost_2)
    - kiszámoltam minden kliens „beleszólásának” mértékét (amount of say) az alábbi képlet szerint:
        - $amount\_of\_say = 0{,}5 * ln\frac{accuracy}{1-accuracy}$
    - ezzel kaptam egy értéket, amely *e* kitevőjében jelent meg. ezt megszoroztam az előző súllyal:
        - $weight = previous\_weight * e^{amount\_of\_say}$
    - az értékeket normalizáltam, hogy az összegük 1 legyen
    - így kaptam a kliensekhez valószínűségeket, amelyeket felhasználva kiválasztottam közülük k darabot (ugyanannyit, amennyi eredetileg is volt)
    - megemlíteném, hogy a nagyobb valószínűségű kliensek többször is kiválasztásra kerülhetnek
    - a szerver ebből a k darab modellből kiszámolta a közös modellt (minden modellt 1/k súllyal számítva)

- Saját algoritmus AdaBoostból merítve – (saját)
    - választottam egy s változót, amelyet felhasználtam az alábbi képletben:
        - $weight = previous\_weight * \frac{accuracy}{1-accuracy}^{s}$
    - azt értékeket normalizáltam, hogy az összegük 1 legyen
    - megemlíteném, hogy az s változó értéke tetszőleges, minél nagyobb értéket adunk meg, annál nagyobb súlyt kapnak a jól teljesítő kliensek
        - de túlságosan nagy értéknél előállhat az a probléma, hogy szinte csak a legjobban teljesítő kliensnek lesz szerepe a súlyozásban
    - az implementációban az s értéke körönként változik, megvizsgálja a szerver, hogy az s melyik értékével lesz a legpontosabb a közös modell
        - hátránya, hogy nagyobb számításigényű
        - egyelőre nem gradiens süllyedéssel van implementálva, hanem egyenletesen választ ki értékeket s-nek (pl. 0.2, 0.4, 0.6, … 1.8, 2.0)

Az első (adaboost) és harmadik (saját) algoritmus alapgondolata hasonló, minden kliens modellje bele fog számítani (valamekkora súllyal) a közös modell kiszámításakor. A második (adaboost_2) algoritmus fő különbsége az, hogy a nagyobb valószínűséget kapó (tehát jobban teljesítő) kliensek többször is kiválasztásra kerülhetnek, és a kisebb valószínűségűek pedig akár egyszer sem.

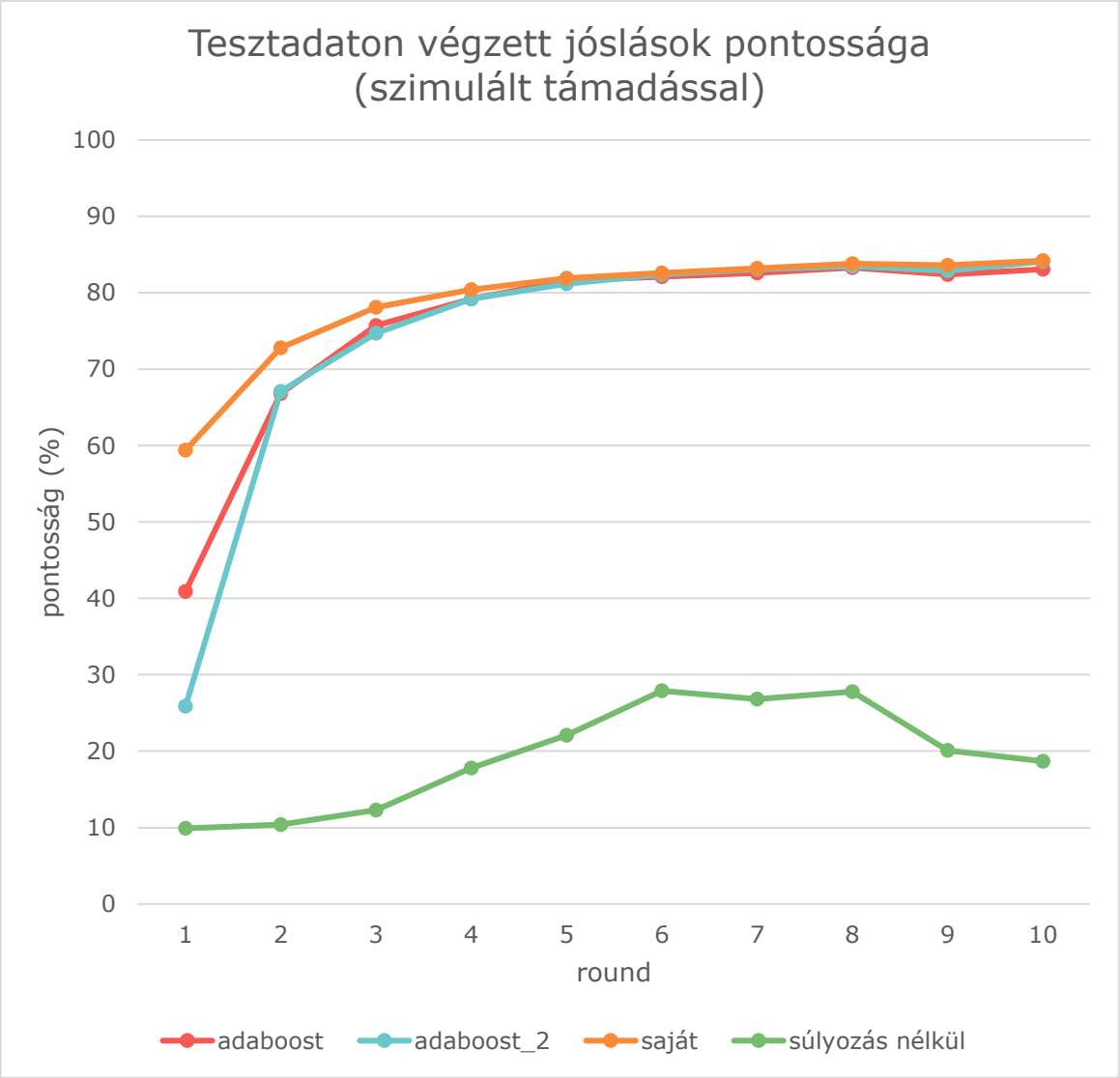
Az első és harmadik algoritmusról továbbá elmondható, hogy a Multiplicative Weights Update metaalgoritmus az alapja. Ennek a lényege, hogy adott egy halmaz (jelen esetben kliensek), melyek között adott egy eloszlás. Minden lépésben az adott i klienshez rendelt valószínűséget megszorozzuk/elosszuk (1+C(i))-vel, ahol C(i) egy valamilyen módon számolt jutalom/büntetés. Ezután szükséges egy normalizálás, hogy az új értékek is a megfelelő eloszlást teljesítsék (jelen esetben az összegüknek egynek kell lennie) [12].

Az algoritmusokat teszteltem, a tesztek során a körönkénti epoch értéke 1, a körök száma 10, az egy körben kiválasztott kliensek száma 10 volt (10%).



A kliensek nagyjából ugyanúgy teljesítenek tekintve, hogy ugyanabból az adatbázisból kaptak ugyanakkora részeket. Emiatt a súlyozás nem hatékonyabb, mint a súlyozás nélküli federált tanulás. Ahhoz, hogy jelentős különbségeket vehessünk észre, szimuláltam egy rosszindulatú támadást. A 100 kliensből 50 kliensnél megváltoztattam a képeikhez tartozó értékeket egy helytelenre (pl. egy 9-es számjegyet tartalmazó képhez 8-ast rendeltem). Ezáltal ezek a támadó kliensek tanuló adatbázisát teljesen elrontottam. A súlyozó algoritmusaink célja, hogy ezeket a rosszindulatú támadásokat kiszűrje.

Az algoritmusokat teszteltem, a tesztek során a körönkénti epoch értéke 1, a körök száma 10, az egy körben kiválasztott kliensek száma 10 volt (10%), a kliensek fele rosszindulatú támadó volt.



Látható, hogy a súlyozás használata nélkül a rosszindulatú támadások nem lettek kiszűrve, és ezáltal a pontosság nagyon kis mértékben tudott csak jobb lenni, mintha tippelne a modell (tudás nélkül 10% eséllyel mondható meg, hogy melyik számjegyről készült a kép). A támadó nélküli esetben is hasonló eredményeket értünk el az algoritmusokkal (~85%), tehát jól működnek erre a kisarkított esetre. A vizsgált algoritmusok közül kezdetben a „saját” teljesített legjobban, de hátránya, hogy nagyobb számításigénye van.

3.3 Adatvédelem biztosítása

Az ismertetett feladat adatvédelmi szempontokból is megvizsgálandó. Az elsődleges célunk az, hogy mindenki csak annyit tudjon, amennyit szükséges tudnia. A problémát meg kell vizsgálnunk a szerver, továbbá a kliensek szempontjából is.

A korábban említett *pontosság* meghatározását úgy kell megoldani, hogy a kliensek ne kapjanak információt arról, hogy hogyan teljesítettek. Ezek alapján egy járható út az, hogy a szerver tesztadatbázisának változóit a frissített modellen letesztelik a kliensek, majd visszaküldik a jósolt eredményeket. Ezeket a szerver összeveti a helyes eredményekkel (amiket csak a szerver tud), és ebből meg is kapja az arányszámokat. Ezt a kódban is implementáltam.

A kliensek minden kör elején tanítják a szervertől kapott modellt, majd egy kör végén az összes kliens titkosítja a kapott modell paramétereit. Ezek tehát titkosítva érkeznek a szerverhez, a szerver pedig (a korábban már tárgyalt súlyozó algoritmussal) ezekből a titkosított paraméterekből kiszámolja a modellt, amely visszaküldésre kerül a kliensek számára (és így bezárul egy kör). A kliensek célja az, hogy a szerver csak és kizárólag az aggregáláshoz szükséges műveleteket tudja elvégezni a kapott adatokon. Ennek megoldásához ismertetném röviden a funkcionális titkosítás alapjait.

A funkcionális titkosítás segítségével egy felhasználó a dekódoló kulcsával megismerheti egy titkosított adat adott függvényét, és semmi mást. Egy megbízható hatóság birtokolja a mesterkulcsot, ezt kizárólag ő ismeri. Ha kap egy f függvényt bemenetként, a mesterkulcsát felhasználva tud generálni egy kulcsot, ami az f-hez tartozik. Akivel ezt a mesterkulcsból származtatott kulcsot megosztja, az képes lesz megismerni bármely adat f függvényét úgy, hogy az adat titkosítva marad.

4. MEGOLDÁSOK ÉRTÉKELÉSE

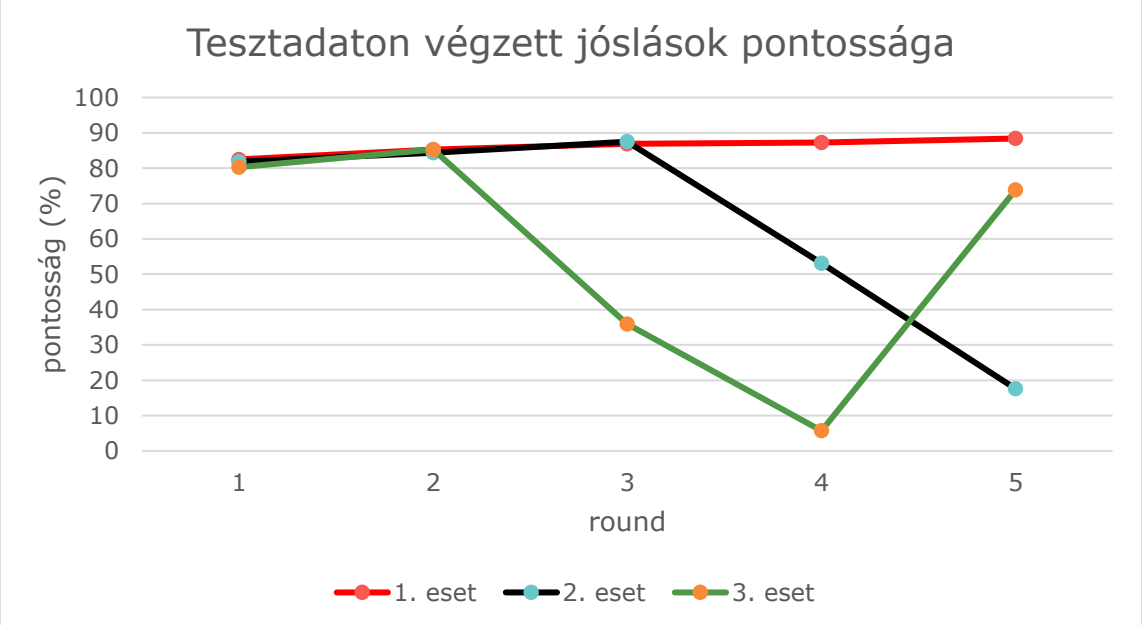
4.1 Federált tanulás

Felmerül a kérdés, hogy egyszerűbben is megoldható-e a körönként frissülő modell kiszámítása. Ahelyett, hogy minden körben az összes kliens modellje részt vesz (valamekkora súllyal) a közös modell számításában, válasszunk ki minden körben csupán egyet (a legjobban teljesítőt), és kizárólag csak az ő általa frissített modellt küldje vissza a szerver a kliensek számára.

Továbbá az is megoldás lehet, hogy a teljesítmény szerint kiszámolt súlyokat valószínűségnek tekintve kisorsolunk egyet a kliensek közül, és csak az ő általa frissített modell kerül elküldésre.

Az utóbbi eset hatékonysági szempontból megkérdőjelezhető, ugyanis előfordulhat az, hogy egy rosszul teljesítő (esetleg rosszindulatú támadó) kliens kerül kiválasztásra, és ez a modell helyességét nagy mértékben leronthatja. Ezt az esetet kódban is implementáltam, és teszteltem arra az esetre, amikor a kliensek fele támadó. A kliensek 10%-a lett körönként kiválasztva, a roundok száma 5 volt. Roundonként a kiválasztott kliensek epochainak száma 10 volt.

Az alábbi eredményeken látszik, hogy futásonként nagy különbség lehet a véletlenszerűen kiválasztott kliensek miatt. A 2. esetben a 4. és 5. körben rosszindulató támadót sorsolt ki az algoritmus, a 3. lefutás során pedig a 3. és 4. körben. Erre a problémára megoldást nyújthat, ha elmentjük a legmagasabb értékű modellt, és csak akkor írjuk felül, ha érkezik egy jobban teljesítő modell.



Mindkét gondolat sérti az adatvédelmi feltételt, ugyanis így amennyiben egy kliens elmenti egy i. körben a szervertől kapott modellt, majd ezt kivonja az i+1. körben kapott modellből, akkor megismerheti azt, hogy az i. kör végén kiválasztott kliens hogyan frissítette a modellt, mik a paraméterei. Megemlítendő, hogy azt nem fogja tudni, hogy melyik kliensről van szó.

Arról nem lesz szó, hogy ez milyen mértékű adatvédelmi problémát okoz, de a fentiek alapján meggyőződhetünk afelől, hogy a problémánk megoldásának tekintetében a felmerült módosítás nem kivitelezhető.

4.2 Kliensek súlyozása

A kliensek súlyozása a jelenlegi helyzetben magától értetődő, ugyanis szélsőséges szituáció volt vizsgálva azokban az esetekben, amikor rosszindulatú támadást szimuláltunk. A kliensek egyik fele kb. ugyanannyira jól teljesít minden körben, a másik fele pedig kb. ugyanannyira rosszul (támadó kliensek). A feladatunk tehát az, hogy a nagyon rosszul teljesítőket kiszűrjük, és csak a nagyon jól teljesítők kapjanak nagyobb súlyokat. Ezt a feladatot a fent említett algoritmusok mindegyike képes elérni. Viszont amennyiben bonyolultabb adatbázissal rendelkező változatosabb pontosságú kliensekkel dolgoznánk, akkor észlelhetnénk jelentősebb különbségeket az algoritmusok teljesítménye között.

Továbbá az algoritmusok egyike sem veszi figyelembe, hogy mekkora méretű adatbázissal dolgoznak az egyes kliensek. Az ebből eredő problémákra jelenthet megoldást, ha kiszabjuk, hogy minimum mekkora adatbázissal kell rendelkeznie egy-egy kliensnek, amelyet egy megbízható felügyeleti szerv ellenőriz.

Fontos megemlíteni, hogy a második (adaboost_2) algoritmusnál esély van arra, hogy egy nagy valószínűségű kliens k-szor legyen kiválasztva, így itt is felmerül az előző pontban ismertetett adatvédelmi probléma. Amennyiben egy kliens elmenti egy i. körben a szervertől kapott modellt, majd ezt kivonja az i+1. körben kapott modellből, akkor megismerheti azt, hogy az i. kör végén k-szor kiválasztott kliens hogyan frissítette a modellt, mik a paraméterei.

5. ÖSSZEGZÉS

A federált tanulás hatékonyságát ellenőriztem, összehasonlítottam a centralizált tanulással. Később megvizsgáltam a helyettesíthetőségét is. Az összes (körönként kiválasztott) kliens helyett csak az egyik kliens kiválasztásával is kellően jó eredményeket értem el, de az adatvédelmi probléma megjelenése miatt – miszerint a kiválasztott kliens általi módosítás könnyen kiszámolható) – elvetettem.

Az adatvédelmi feltételeket csak az egyik oldalról sikerült biztosítani. A szerver szempontjából privátnak tekinthető tesztadatbázis értékei nem jutnak el a kliensekhez, csak a változókat kapják meg. Viszont a kliensek bizalmas adatainak titkosítása a program jelenlegi formájában nincs megvalósítva, ezt a jövőben szükséges lesz implementálni az elvárt működéshez.

A kliensek súlyozását pontosság alapján sikerült megvalósítani. Ismertettem 3 algoritmust, melyek a tesztelésre kialakított rendszerben (100 kliens MNIST adatbázis feldarabolásával) jól működnek. A rosszindulatú támadás szimulációjánál is elegendő mértékben kiszűrte a támadókat. Fontos kiemelni, hogy ebben az esetben viszonylag egyszerű feladat volt elkülöníteni a „jó” klienseket a „rosszaktól”. A „csak egy kliens kiválasztásánál” említett adatvédelmi probléma felmerülhet a második algoritmusban, emiatt az első és a harmadik algoritmus tekinthető megbízhatóbbnak. Ennek a kettő algoritmusnak ugyan az az alapvető lényege, egy képlet segítségével (melyben megjelenik a pontosság mint változó) megtörténik a súlyozás, majd normalizálás után ezekkel a súlyokkal a szerver által megtörténik a közös modell számítása.

6. JÖVŐBELI MUNKA

A probléma egyik feltétele volt, hogy a kliensek adatvédelmét biztosítsuk. Az elméleti kidolgozása és a kód implementálása szükséges, a dokumentumban csak néhány, a megoldás alapjaihoz szükséges információkat ismertettem.

Az alábbiakban kigyűjtöttem a probléma során felmerülő kérdéseket is, melyekre érdemes lehet választ találni.

- *Múltbeli teljesítmény szerepének mértéke a súlyozásnál*
- *A kliensek adatbázisméretének beépítése a súlyozó algoritmusba*
- *További adatbázisokkal, beállításokkal a súlyozó algoritmusok tesztelése (pl. különböző adatbázis méretek, MNIST helyett más adatbázis)*
- *A saját algoritmusban s értékének gradiens süllyedéssel való meghatározása*

7. IRODALOMJEGYZÉK

[1] „The Multiplicative Weights Update Method: a Meta Algorithm and Applications,” [Online]. Available: https://www.cs.princeton.edu/~arora/pubs/MWsurvey.pdf. [Hozzáférés dátuma: 25 május 2020].

[2] „AdaBoost - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/AdaBoost. [Hozzáférés dátuma: 23 május 2020].

[3] „Communication-Efficient Learning of Deep Networks,” [Online]. Available: https://arxiv.org/pdf/1602.05629.pdf. [Hozzáférés dátuma: 23 május 2020].

[4] „ElGamal - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/ElGamal_encryption. [Hozzáférés dátuma: 23 május 2020].

[5] „Federated Learning,” [Online]. Available: https://federated.withgoogle.com/. [Hozzáférés dátuma: 23 május 2020].

[6] „Forget Homomorphic Encryption - Here Comes Functional Encryption,” [Online]. Available: https://research.kudelskisecurity.com/2019/11/25/forget-homomorphic-encryption-here-comes-functional-encryption/. [Hozzáférés dátuma: 23 május 2020].

[7] „Functional Encryption,” [Online]. Available: https://irandoc.ac.ir/sites/fa/files/attach/event/boneh2012.pdf. [Hozzáférés dátuma: 23 május 2020].

[8] „MNIST,” [Online]. Available: https://syncedreview.com/2019/06/19/mnist-reborn-restored-and-expanded-additional-50k-training-samples/. [Hozzáférés dátuma: 23 május 2020].

[9] „Privacy-preserving Weighted Federated Learning,” [Online]. Available: https://arxiv.org/pdf/2003.07630.pdf. [Hozzáférés dátuma: 23 május 2020].

[10] „Simple Functional Encryption Schemes,” [Online]. Available: https://eprint.iacr.org/2015/017.pdf. [Hozzáférés dátuma: 23 május 2020].

[11] Caltech, „Learning from Data (MOOC),” [Online]. Available: https://work.caltech.edu/telecourse.html. [Hozzáférés dátuma: 23 május 2020].

[12] A. Gupta, „How Federated Learning is going to revolutionize AI,” [Online]. Available: https://towardsdatascience.com/how-federated-learning-is-going-to-revolutionize-ai-6e0ab580420f. [Hozzáférés dátuma: 23 május 2020].

8. MEGJEGYZÉS

A mesterséges intelligencia alapjainak elsajátításához a Caltech online elérhető MOOC (Machine Learning online course) videóanyagát használtam.

A fejlesztői környezet létrehozásakor számos probléma merült fel (pl. telepítési problémák ékezetes elérési út miatt), viszont a Jupyter Notebookon alapuló Google Colaboratory segítségével pillanatok alatt felállítható a rendszer. Előnye, hogy tartalmazza a Keras, TensorFlow és más népszerű könyvtárakat is, továbbá nem használja a számítógép erőforrásait, ugyanis a Google által biztosított távoli gépen történnek a számítások.

A félévi munkám alapvetően négy részre osztható:

1. *Az első néhány hét során az alapok elsajátítása volt a feladatom, a mesterséges intelligencia alapjainak elsajátításával foglalkoztam.*
2. *Megismerkedtem a gépi tanulás egyik technikával, a federált tanulással.*
3. *A funkcionális titkosítás alapjaival ismerkedtem meg.*
4. *A kliensek súlyait határoztam meg teljesítményük alapján többféle algoritmus segítségével.*
5. *Dokumentáltam az elért eredményeket.*