Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Botond Ákos

TDK dolgozat

BÖNGÉSZÉSI ÉLMÉNY BECSLÉSE GÉPI TANULÁSI MÓDSZEREKKEL

KONZULENSEK:

Megyesi Péter

Barta Gergő

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló

A mai hálózatok túlterheltsége miatt az Internet szolgáltatóknak egyre fontosabb elemeznie, hogy a felhasználóik milyen minőségi paraméterekkel töltenek le bizonyos tartalmakat. Mivel a szubjektív felhasználói élmény elemzése túl időigényes, ezért a gyakorlatban leginkább a forgalmi mintákból becsülhető kvantitatív adatokra hagyatkoznak. Az utóbbi időben a hagyományos elemzési módszerek egyre kevésbé tudják tartani a lépést a folyamatosan változó webes forgalommal, így a kutatók a sok területen sikerrel alkalmazott gépi tanulási módszerek felé fordultak. A gépi tanulási módszerek előnye, hogy használatukkal akkora adathalmaz feldolgozására nyílik lehetőségünk, amelyet klasszikus módszerekkel szinte lehetetlen lenne kezelnünk. Ezek a különböző algoritmusok már bizonyítottan alkalmasak videó nézés közbeni QoE (Quality of Experience) mérésre és hálózati forgalom azonosítására. Ezen kutatások alapján úgy gondolom, hogy a gépi tanuló algoritmusok sikerrel alkalmazhatók oldal betöltési idők előrejelzésére is.

A dolgozatomban bemutatom az általam fejlesztett mikrokörnyezetet, amelyben Google Chrome-mal böngészve weboldalak hálózati paramétereit tudjuk mérni és hozzárendelni forgalmi mintákhoz. Az így kapott hálózati paraméter és forgalmi minta párosból létrehoztam egy saját adatbázist, amelyet gépi tanuló algoritmusokkal elemeztem. A betanított algoritmus már alkalmas egy web letöltésfolyam első néhány csomagjának paraméterei alapján megbecsülni, hogy az adott felhasználó elégedett-e a weboldal betöltési sebességével. A létrehozott keretrendszer másik nagy előnye, hogy a gépi tanulásnak köszönhetően már azelőtt rendelkezésünkre állhatnak a szükséges adatok, hogy a weboldal betöltése valójában befejeződött volna, ugyanis az elemzéshez nincs szükségünk az adott folyamban lévő csomagok összességére.

Az elkészült keretrendszerrel a szolgáltatók a felhasználó közreműködése nélkül juthatnak információhoz a hálózati paraméterekről, így a lehetséges problémáról hamarabb értesülnek, illetve a többlet információt felhasználhatják a hálózat optimalizálásához.

Abstract

Due to the overload of today's networks, it is more and more important for the internet service providers to analyse the quality parameters of user experience. Since the measurement of subjective user experience is too time-consuming, they rely on quantitative data estimation based on traffic patterns. Recently, conventional analysis methods cannot keep up with the ever-changing web traffic thus researchers turned to Machine Learning (ML) methods which was proven to work successfully in many areas. The advantage of ML methods is by using them we have the possibility to process big data which would be almost impossible using traditional methods. Recent literature showed that ML algorithms are suitable for measuring Quality of Experience (QoE) of online video watching and also for network traffic identification. On the basis of these researches, I believe that ML algorithms can be used successfully for predicting the page load times as well.

I developed a framework in which websites download parameters can be measured by automatically browsing with Google Chrome while the network traffic is captured. I created my own database using the website download parameters and the associated traffic patterns and then I analysed them with ML algorithms. The trained algorithm is suitable to estimate whether the user is satisfied with the loading speed of the website based on the first few packets in the traffic stream. The other important advantage of the created framework is that we can obtain the necessary estimation earlier than the loading of website is completed. We can achieve this because we don't need to analyse all the packet of a certain web flow.

Using the created framework, service providers can obtain information about network parameters without user interaction, thus they can get information earlier about the potential problems and they can use this information to optimize their network.

1. Bevezetés

1.1 Forgalomelmélet és szolgáltatás minőség mérése

A forgalomelmélet már a XX. században is fontos kérdés volt távközlési hálózatok tervezésénél. Agner Krarup Erlang (1878-1929) dán matematikus volt az első, aki megfogalmazta a matematikai problémát. 1909-ben publikált cikkében állította fel elméletét. Az elmélet a telefonhálózatok fejlesztésével párhuzamosan fejlődött, és lényegi elemévé vált a klasszikus távközlési hálózatok tervezésének [1].

A hálózati forgalom mérésére először a klasszikus telefonhálózatok tervezésekor mutatkozott igény. A telefonhálózatok forgalmának statikus jellege tette lehetővé, hogy olyan "univerzális törvényeket" találjunk, mint a hívások keletkezésének Poisson természete [1]. Ezt a közelítést megtehettük, mert ez a leegyszerűsített modell nagyon jól használható volt a tervezés folyamatában. Később azonban a hálózatok egyre bonyolultabbá váltak és már nem csak beszédátvitelre, hanem faxra és Internet elérésre is használni kezdték.

Manapság a forgalom mérése mellett a szolgáltatás minőségének mérése is egyre hangsúlyosabbá válik. Korábban a szolgáltatók a hálózatuk minőségét bizonyos hálózati paraméterek függvényében mérték. Ezeket a paramétereket QoS (Quality of Service) paramétereknek nevezzük. A tapasztalat azonban azt mutatja, hogy a QoS érték és a felhasználói élmény nem minden esetben egyezik meg. Például hiába van csomagvesztés egy előre meghatározott fix érték alatt, az adott felhasználó böngészési élménye még lehet, hogy nem megfelelő. Az ehhez hasonló anomáliák megszüntetésére bevezettek egy másik mérőszámot a QoE-t (Quality of Expreience). A QoE mérése már nem fixen előre beprogramozott mérőszámok szerint működik, hanem inkább a böngészési élményt írja le a felhasználó szemével. Azonban ennek a paraméternek az automatizált mérése nem egyszerű, legtöbbször felhasználói interakciót igényel.

1.2 Feladatom bemutatása

A TDK dolgozatomban egy olyan keretrendszert mutatok be, amellyel a megfelelő tanító mérések elvégzése után a szolgáltatók képesek lesznek ügyfeleik weboldal betöltési idejét gépi tanuló algoritmusok segítségével előre megjósolni az első pár száz csomag alapján. Ezzel a megoldással, hogy nem kell az összes csomagot megvizsgálni, időt és erőforrást takarítanak meg, valamint megfelelő automatizmusok esetén akár az adott letöltés folyamban is beavatkozva, az ügyfélnek nagyobb prioritást adva a böngészési élményt képesek lehetnek javítani, mielőtt az ügyfél számára egyáltalán feltűnt volna a böngészési paraméterek romlása.

Az elkészült rendszerrel lehetőségünk nyílik QoE mérésére is felhasználói közbeavatkozás nélkül. A rendszert használva a szolgáltatóknak nincs szükségük felhasználói visszajelzésre, a forgalmi adatokból automatikusan látják a hálózati hibákat, amelyek kijavítását azonnal megkezdhetik a felhasználó közreműködése nélkül.

1.3 További fejezetek tartalma

A TDK dolgozat további részében bemutatom, hogyan készült el a követelményeknek megfelelő keretrendszer. A második fejezetben egy rövid áttekintést olvashatunk a dolgozat elkészítésének motivációjáról és a hasonló tudományos munkák megoldásairól. Ezek után a 3. fejezetben ismertetem a munkám során felhasznált eszközöket. Ezt követően a 4. fejezetben részletesen bemutatom a megvalósított keretrendszert és annak működését, majd az 5. fejezetben szemléletes diagramok segítségével ismertetem az elért eredményeket. Végezetül a 6. fejezetben összefoglalom a munkámat, és kitérek az esetleges jövőbeli továbbfejlesztési lehetőségekre is.

2. Motiváció és kapcsolódó tudományos kutatások

Manapság a webböngészés a mindennapok elengedhetetlen részévé vált. A felhasználóknak számtalan eszköz áll a rendelkezésükre, amelyeken képesek a világhálót böngészni és ezek az eszközök változatos módokon kapcsolódnak az internetre. Azonban mindegy, hogy vezetékes internetet, Wi-Fi-t, vagy mobil hálózatot használunk a böngészéshez az oldalak betöltési ideje egy minden platformon értelmezhető mennyiség. A böngészési élmény ezzel az értékkel írható le a legjobban. Az oldal betöltési időt azonban nem lehet önmagában értékelni, tudnunk kell, hogy az a betöltési idő milyen weboldalhoz kapcsolódott. Ennek a két adatnak a függvényében már reális képet kaphatunk a felhasználó böngészési élményéről.

Az oldal betöltési idő témakörében számtalan tudományos cikk született. Ezen kutatások egy része az oldal betöltési idők csökkentésére irányul és az oldal betöltési idő csökkentése által okozott bevétel váltakozást vizsgálja. Például az Amazon bevétele 1%-kal növekszik, ha az oldal betöltési idejük 0,1 másodperccel csökken [2]. Az ilyen konkrét közvetlen gazdasági előnyt mutató példák miatt a hálózat tervezők folyamatosan az oldal betöltési idők csökkentésén dolgoznak. Ezt a legegyszerűbben CDN-ek (Content delivery network-ok) használatával érhetik el, de létezik szcenárió, amely úgy csökkentené az oldal betöltési időket, hogy napközben az üzleti weboldalaknak, éjszaka pedig a szórakoztató weboldalaknak adna nagyobb prioritást [3].

Az előző bekezdésben foglaltak alapján kijelenthetjük, hogy az oldal betöltési idő a böngészési élmény kulcsmetrikája. Azonban ennek mérése a szolgáltatói oldalon komoly kihívás. Ennek a megoldására fejlesztettem ki a saját keretrendszeremet, amely előzetes gépi tanítás után, erőforrás és idő takarékosan a csomagok töredékének vizsgálatával képes megállapítani ezt a kulcsmetrikát. A keretrendszer megvalósításához korábbi tudományos munkák adták az ötletet. A gépi tanulási módszereket már sok hálózati forgalom mérésével kapcsolatos területen sikerrel alkalmazták. Ezek közül párat emelnék ki.

A hálózati forgalom egy részét VoIP azaz Voice Over IP hangforgalom adja. A legújabb mobilhálózatok is már egyre gyakrabban használják ezt a technikát. Azonban az ilyen módon felépített kapcsolat, és a kapcsolat minősége klasszikus módszerekkel nehezen mérhető. Azonban egy kutatás szerint bizonyos hálózati paraméterek vizsgálatával a kapcsolat minősége megjósolható [4]. Ehhez teszt méréseket kellet végezni, ahol kontrollált körülmények között és a hálózati paraméterek rögzítése mellett ellenőrzött tanítással gépi tanuló algoritmusok képesek voltak mintákat felfedezni, amely alapján végül 75%-os pontossággal prediktálható volt a kapcsolat minősége.

A webes forgalom egyik legnagyobb részét a videó streamek továbbítása jelenti. Itt is kulcsfontosságú a QoE mérése, ugyanis ezeknek a streaming megoldásoknak a többségénél a videó több felbontásban is elérhető, és a hálózati paraméterektől függ, hogy az automatikus rendszer milyen minőségű és felbontású videót küld az eszközünkre. Ezért itt is nagyon fontos a hálózat paraméterek mérése és becslése, amelyhez szintén sikerrel alkalmaztak gépi tanuló algoritmusokat [5][6].

Más kutatások pedig a mobil hálózatok QoE becslésére használják sikerrel a gépi tanuló algoritmusokat [7], vagy mobil alkalmazásokhoz fejlesztenek QoE mérésére alkalmas keretrendszert [8].

A kutatások másik irányzata pedig IP csomagok osztályozására irányul [9]. Itt is gépi tanulás felhasználásával vizsgálják a hálózati adatforgalmat és utána osztályozó algoritmusok segítségével osztályokba sorolják az adatforgalmat. Ennek a folyamatnak mindennapokban tapasztalható gyakorlati jelentősége is van. Egyes mobilszolgáltatóknál, köztük a Telekomnál vannak tematikus mobilinternet csomagok, amelyekben külön-külön elő lehet fizetni korlátlan böngészésre, közösségi oldalak használatára vagy akár korlátlan TV nézésre is. Ezen csomagok használatakor a szolgáltató a forgalmi mintákból becsüli meg, hogy melyik alkalmazáshoz tartoznak, és ez alapján a mobilinternet keret vagy a korlátlan tematikus opcióhoz kapcsolódik a leforgalmazott adatmennyiség. Az osztályozás másik gyakorlati jelentősége a torrentezés szűrése.

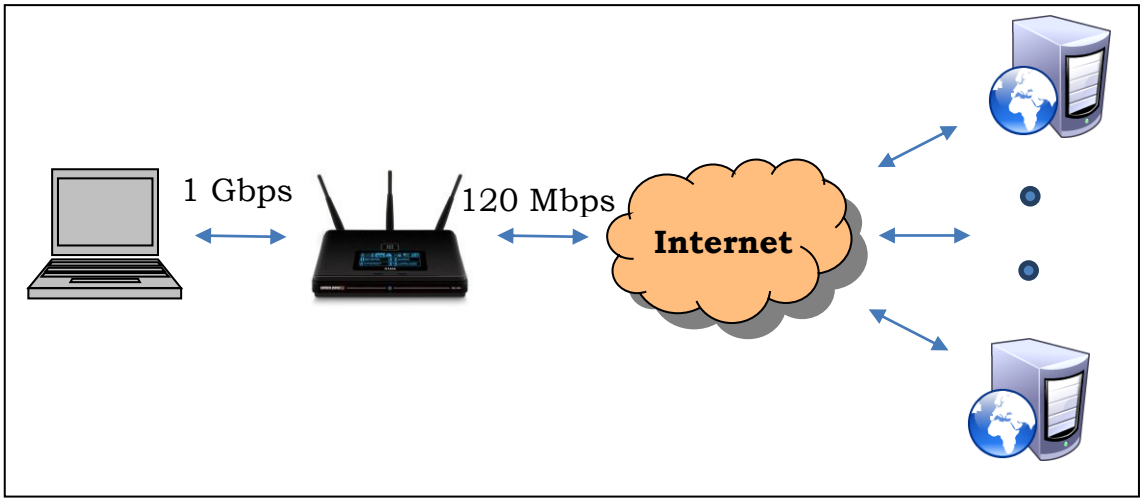
Az eddig bemutatott kutatások adták az ötletet, egy gépi tanulási algoritmust felhasználó, böngészési QoE becslő implementálására. A gépi tanuló algoritmusokat már sok helyen felhasználták, azonban weboldal betöltés idő becslésére korábban még nem, pedig a kapcsolódó munkák alapján megfelelő leírókat találva, sikeresen lehet egy ilyen rendszert kiépíteni, amelyet számtalan később részletesen leírt gyakorlati probléma megoldására fel lehet használni.

3. Analízis eszközök

3.1 Mérési elrendezés

A legpontosabb mérőkörnyezet előállításához létrehoztam egy új Google Chrome profilt, amelyet csak a mérésekhez használtam, így az egyébként használt böngésző kiegészítőim nem zavarták a mérést. A pontosság érdekében töröltem minden böngészési adatot, és a fejlesztői menüpontban kikapcsoltam a cache (magyarul: gyorsítótár) használatot is. A méréseket inkognitó módban folytattam, így a böngészési adatok nem kerültek mentésre, így az oldalak többszöri betöltésekor is minden adatot a szerverről gyűjtöttem be.

A méréseket vezetékes internetkapcsolattal végeztem, melynek maximális sebessége 120 Mbps. A mérések során ezt a sebességet változtatgattam, amelyet a router menüjéből tudtam megtenni.



3-1. ábra Mérési elrendezés

3.2 HAR fájlok

A HAR elnevezés a HTTP Archive szókapcsolat rövidítéséből ered [10]. Ez a formátum a legelterjedtebb HTTP kérések követésére. A böngésző által betöltött összes objektumot elmenti, beleértve ezek betöltési idejét, formátumát és a forrás helyét is. Az elmentett adatokat JSON dokumentumként tárolja, amely egy alacsony szintű tároló formátum.

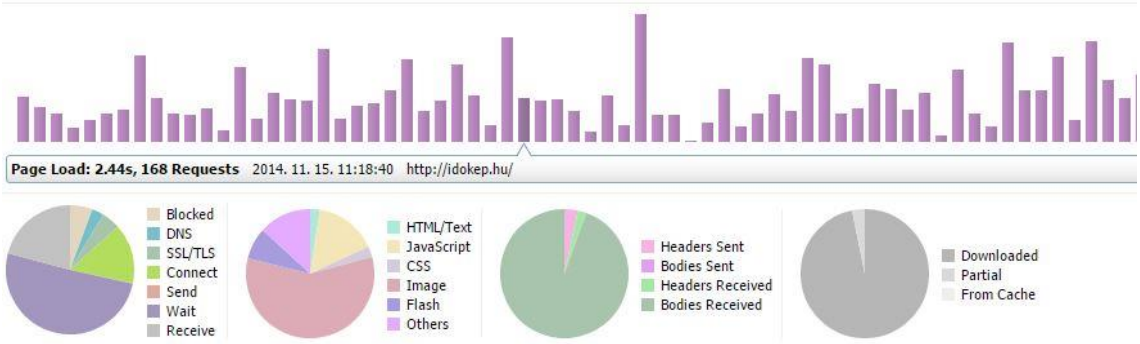
3.2.1 HAR fájlok létrehozása

A HAR fájlok elmentésére a Google Chrome gyárilag alkalmas. A fejlesztői eszközök menüpont segítségével hoztam létre a később feldolgozandó HAR fájlokat. Első lépésként az automatizáló programom beállította a cache használat tiltását és a log fájlok megőrzését, majd elkezdte a böngészést a weboldalon. A böngészés befejezése után pedig a Chrome-ból automatikusan elmentette a HAR fájlt a későbbi feldolgozáshoz. Az automatizálás folyamatát a 3.4-es fejezetben fogom kifejteni részletesebben.

3.2.2 HAR fájlok ellenőrzése

A létrehozott HAR fájlokat a feldolgozás előtt egy webes HAR Viewer-rel [11] vizsgáltam meg, valamint a feldolgozó programom működését is ennek a segítségével ellenőriztem. A weboldal minden adatot kiolvas a HAR fájlból és a betöltési időket diagramokon is ábrázolja.

A betöltési idők alatt látható kördiagramokon egyéb adatokat is automatikusan ábrázol a program. A négy kördiagram közül az utolsót emelném ki, amelyről leolvasható, hogy cache-ből nem történt betöltés egyik oldalnál sem.



3-2. ábra HTTP Archive Viewer

3.3 PCAP fájlok

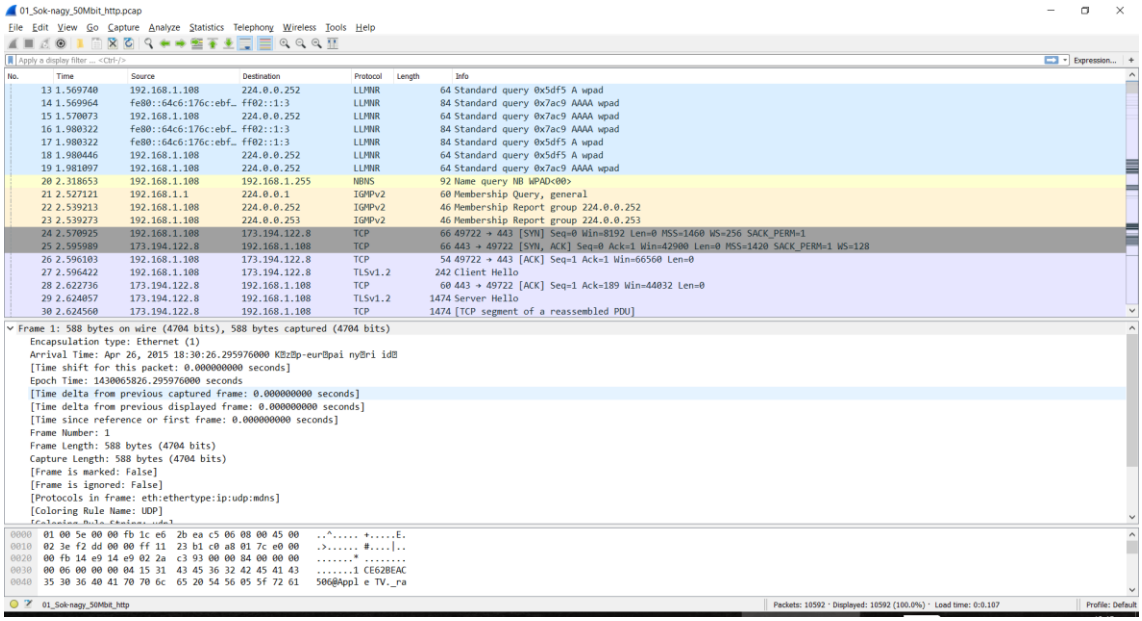
A PCAP fájl elnevezés az angol **P**acket **CAP**ture szókapcsolat rövidítéséből ered [12]. Hálózati forgalmi adatokat naplózhatunk vele későbbi feldolgozás céljából.

3.3.1 PCAP fájlok létrehozása

A PCAP fájlok létrehozásához a windump [13] nevű programot használtam. Minden oldal letöltést külön PCAP fájlban mentettem el. Ennek a folyamatnak az automatizálásához egy AutoIt-ben általam megírt kódsort használtam, amelyet a 3.4-es fejezetben fejtek ki részletesebben.

3.3.2 PCAP fájlok ellenőrzése

A PCAP fájlokat legegyszerűbben a Wireshark nevű programmal ellenőrizhetjük, amely grafikus felületen több színnel kiemelve mutatja a különböző csomagokat, valamint több összesítő nézet is található benne [14].



3-3. ábra Wireshark

3.4 AutoIt

Az AutoIt egy grafikus automatizáló program, amelynek használatával gyakorlatilag bármilyen statikus feladatot automatizálhatunk, így a böngészés elvégzéséhez is alkalmas [15].

Az AutoIt-ben megírt makróm segítségével egy kattintással elindíthatjuk a méréseket, amely utána automatikusan elvégzi az alábbi feladatokat:

- böngésző megnyitása
- inkognitómód aktiválása
- fejlesztői mód aktiválása
- cache használat tiltása és log fájlok megőrzése
- routerbe belépve sávszélesség megváltoztatása
- PCAP fájlok mentéséhez windump program automatikus nyitása, oldal betöltése után pedig zárása
- HAR fájlok mentése

A fenti automatizálási lépéket ciklusba foglalva különböző sávszélességek melletti mérések is egy kattintásra elindíthatóak. A program használata nélkül szinte lehetetlen lett volna a gépi tanulás működéséhez szükséges mennyiségű böngészési adat előállítása.

3.5 Perl

A HAR és PCAP fájlok feldolgozásához a Perl nyelvet választottam, ugyanis korábbi munkáim, előtanulmányaim eredményeképpen már jól ismerem ezt a nyelvet.

A Perl nyelvet Larry Wall-nak köszönhetjük, aki 1987-ben kezdte el fejleszteni és még ezen év decemberében meg is jelentette a Perl első kiadását. Azóta folyamatosan fejlesztik a nyelvet, ami jelenleg az 5-ös verziónál tart.

3.5.1 Reguláris kifejezések

A Perl nyelv egyik legnagyobb előnye a nagyon fejlett mintaillesztési lehetőség, vagyis a reguláris kifejezések kezelése. Ezek támogatását a PCRE könyvtár végzi. Erre a képességre nagy szükségem volt a HAR és PCAP fájlok feldolgozása során. Például reguláris kifejezés használatával lehet megtalálni a PCAP fájlokban, hogy melyik az első csomag, amely már a mérni kívánt oldal letöltéséhez kapcsolódik.

3.5.2 Felhasznált CPAN modulok

A CPAN a Comprehensive Perl Archive Network (kiegészítő Perl archív hálózat) rövidítése. Ezen a weblapon több mint 100 000 Perl nyelven írt program és dokumentáció található [16]. Ez a hatalmas adatbázis a Perl nyelv népszerűségének köszönhetően jöhetett létre.

- Archive::HAR [17]

  Ennek a modulnak nagy szerepe volt abban, hogy a feldolgozáshoz a Perl nyelvet választottam, ugyanis ennek a segítségével érhetem el a HAR fájlban tárolt adatokat.

- Net::TcpDumpLog [18]

  A modul meghívásakor betöltődik a PCAP fájl tartalma a memóriába a további feldolgozás céljából.

- NetPacket::Ethernet/IP/TCP/UDP [19]

  A PCAP fájlok feldolgozásához nagy segítséget jelentett ez a modul, ugyanis ennek segítségével tudtam dekódolni a PCAP fájlok tartalmát.

3.6 Gépi tanulás

A gépi tanuló algoritmusok futtatásához a Perl helyett a Python nyelvet választottam, amelyhez számtalan gépi tanulást és adatelemzést lehetővé tevő függvénykönyvtár érhető el. Ezek közül én a legnépszerűbbet a „scikit-learn” [20] nevű eszköztárat használtam. Ezen az eszköztáron kívül a „pandas” nevű adatkezelő könyvtárat használtam [21].

Manapság nagy népszerűségnek örvendenek a gép tanulási (ML - Machine Learning) módszerek. Segítségükkel olyan összefüggéseket ismerhetünk fel az adatokban, amelyek felismerésére hagyományos módszerekkel nem nyílna lehetőség.

A gépi tanulás a mesterséges intelligencia egy fajtája. Különlegessége, hogy képes tapasztalatokból tudást generálni. Felépítése hasonlít az emberi tanuláshoz és következtetéshez. Tanítása azonban körültekintés igényel, ugyanis a legjobb eredmény eléréséhez nagyon pontosan kell definiálnunk a kérdést, és a megoldáshoz segítő releváns információknak is rendelkezésre kell állniuk. Az algoritmus működése során nem csupán megtanulja a konkrét mintákhoz tartozó értékeket, hanem mélyebb összefüggéseket és szabályokat keres, amelyeket később alkalmaz. Az oldal betöltési idők előrejelzéséhez nagy szükségem volt a gépi tanulás alkalmazására, ugyanis gépi tanuló algoritmusok használata nélkül a kitűzött feladatom is megoldhatatlan lett volna.

A konkrét esetben két alapjaiban különböző megoldási módszer közül választhattam. Az első elképzelések szerint egy **osztályozási feladatot** oldottam volna meg, amely egy konkrét határt kijelölve 4 felé osztja az eredményeket.

- rövid betöltési idő a valóságban – helyesen prediktálva
- hosszú betöltési idő a valóságban – helyesen prediktálva
- rövid betöltési idő a valóságban – helytelenül prediktálva
- hosszú betöltési idő a valóságban – helytelenül prediktálva

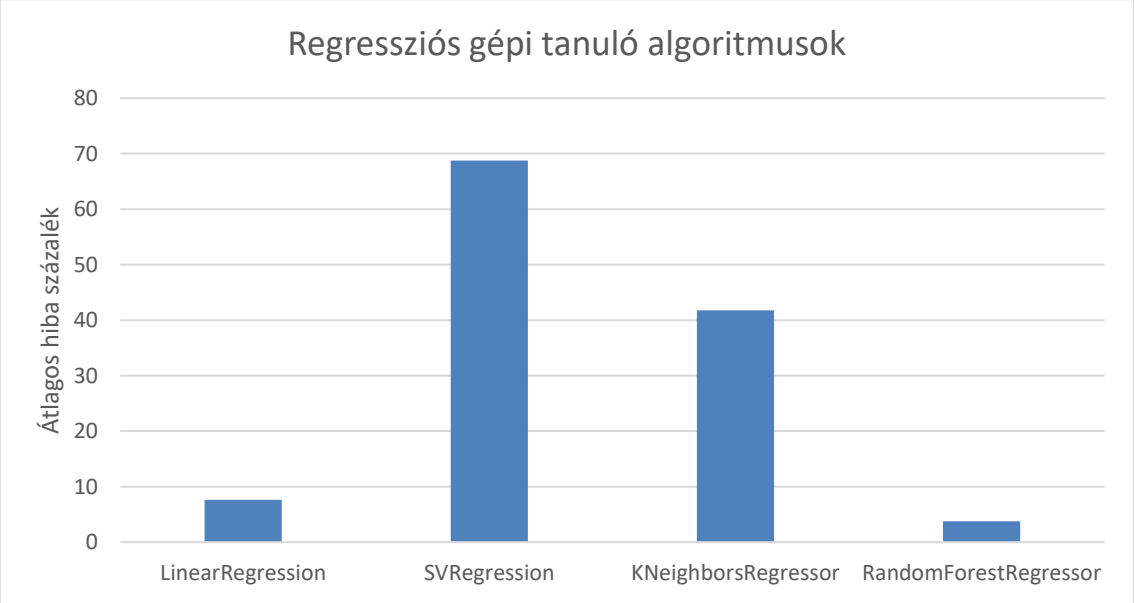
Ezt a modellt azonban elvetettem, mert a letöltési sebesség változtatásából adódó betöltési időváltozásokat nem tudta volna a modell helyesen kezelni. Kisebb sebességnél más számít rövid betöltési időnek, mint nagyobb sebességnél, valamint a kapott eredmények is nagyban függtek a szubjektívan meghatározott betöltési idő limittől. A limit alatt lévő betöltési időket rövidnek, a felette lévőket hosszúnak neveztem.

Az elkészült keretrendszerben egy másik megoldást, **regressziót** használtam. Regressziót használva a gépi tanuló algoritmus egy konkrét kvantitatív becslést ad az oldalbetöltés idejére, amelyet már össze tudtam vetni az általunk ismert eredményekkel, így ellenőrizve a modellünk helyességét. Ez azonban már a kiértékeléshez tartozik, melyet az 5. fejezet tartalmaz.

A regressziós megoldást kijelölve is még sok lehetséges algoritmus közül választhattam. Ezek közül csak párat emelnék ki, amelyekkel próbálkoztam.

- Linear Models - Lineáris modellek
- Support Vector Machines - SVM - Tartóvektor-gépek
- Nearest Neighbors - Legközelebbi szomszédok
- Ensemble Methods - Együttes módszerek

A következő 3-4. ábrán a fenti 4 algoritmus átlagos hiba százalékait ábrázolom. A feldolgozás során kapott adatsort és értékeit nem módosítottam, a függvényeket alapbeállítások mellett, paraméterek hangolása nélkül hívtam meg.

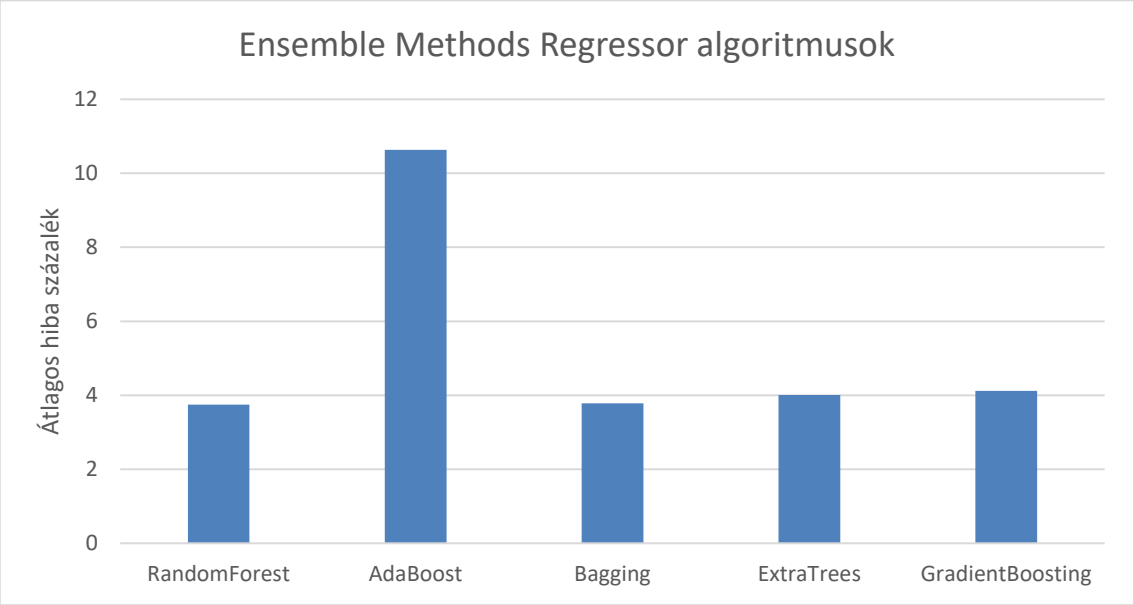


3-4. ábra Regressziós gépi tanuló algoritmusok

A 3-4. ábráról leolvashatjuk, hogy az adott feladatra, melyik algoritmus milyen átlagos hiba százalék eredményt adott. A legalacsonyabb hiba százalékot az Ensemble Methods algoritmusok közé tartozó Random Forest Regressor (Véletlen Erdő Regresszió) algoritmus használatával értem el, így a továbbiakban az Ensemble Methods alapú algoritmusok között kerestem a feladatra legmegfelelőbb algoritmust.

3.6.1 Ensemble Methods algoritmusok

Az Ensemble Methods algoritmusok, olyan komplex technikák, amelyek több egyszerű modell kombinálásával tudnak pontos eredményeket szolgáltatni. A legjobb algoritmus keresésének következő lépése a különböző Ensemble Methods algoritmusok összehasonlítása. Ezt az összehasonlítást a 3-5. ábrán láthatjuk.



3-5. ábra Ensemble Methods Regressor algoritmusok

A 3-5. ábra alapján az AdaBoost algoritmus kivételével az algoritmusok hasonló eredményeket adnak. A hasonló eredmények közül a választásom a Random Forest algoritmusra esett, ugyanis egyszerűsége miatt számítási kapacitást spórolhatunk meg, és bár minimális különbséggel de ennek az algoritmusnak a használata eredményezte a legkisebb hiba százalékot.

3.6.2 Random Forest Regressor

Az előzetes mérések alapján a Random Forest gépi tanuló algoritmust választottam a méréseimhez. Ebben a fejezetben az algoritmus működését és tulajdonságait fogom kifejteni [22].

A Random Forest algoritmus több döntési fát használ, amelyek csak részben különböznek egymástól. Az algoritmus pontossága függ az egyes fák erősségétől és a fák között lévő korrelációtól.

A fák felépítésének módja a következő. A tanító esetek számát N-nek, a változók számát pedig vegyük M-nek. Az egyes döntésekhez felhasznált változók száma, pedig m. Fontos, hogy az egyes döntésekhez csak az összes változó számánál sokkal kevesebb változót vegyünk figyelembe, így több döntési fa építhető kicsi korrelációval (képletben: $m \ll M$). A tanító eseteket kétfelé bontva, n tanító eset és $N - n$ tesztelő eset lesz. A fa minden egyes csomópontján véletlenszerűen választunk m változót, amely alapján meghozzuk a döntést. Ezek alapján az m változók alapján számítjuk ki a legjobb vágásokat. A lépések ismétlésével felépíthető a meghatározott mennyiségű döntési fa.

Összefoglalva, egy adott fa felépítésénél a megfigyeléseket és az attribútumokat is véletlenszerűen választom. Ez okozza az erdő véletlenszerűségét és biztosítja, hogy iterációnként eltérő fák épüljenek.

Az erdő hatékonyságát a következő paraméterekkel befolyásolhatjuk:

- n_estimators – az erdőben lévő fák számát adja meg
    - több fától javul az eredmény, de csak bizonyos pontig
    - több fánál a számítási igény is növekszik
- max_depth – a fák maximális mélysége
    - nagyobb mélységtől javul az eredmény, de nő a túltanulás veszélye

A Random Forest algoritmus alkalmazásának nagyon sok előnye van. Az algoritmusnak a többi algoritmushoz viszonyítva gyors futási ideje van, amely a feladatomban fontos tényező. Valamint jól skálázható és robosztus, amelyek szintén kritikus tényezők. A futása jól párhuzamosítható, amely egy komolyabb rendszer kiépítésénél fontos szempont. Különlegessége, hogy a változókat fontossági sorrendbe tudja állítani, így a fontosnak jelölt változók alapján további ahhoz hasonló, de nem korrelált változók létrehozásával tovább tudtam javítani az algoritmus pontosságát. Futás közben a fontos változókat a gyökér közelében teszteli, míg a kevésbé fontosokat a levelekhez közelebb, így jobb eredményt elérve.

4. Implementálás

4.1 Weboldal meghatározása

Az implementálás első lépése a mérendő weboldal meghatározása volt. A weboldal kereséshez az egyik fontos szempont az volt, hogy elég nagy méretű legyen ahhoz, hogy a rendelkezésre álló sávszélességet legalább közelítőleg kihasználja. A másik fontos szempont a jó minőségű kapcsolat volt, vagyis az oldal betöltési idő hasonló körülmények között ne ingadozzon szélsőségesen. Ezeket a megkötéseket azért alkalmazom, hogy a keretrendszert valós teljesítményét jobban tudjam mérni és a rendszert tovább tudjam fejleszteni úgy, hogy később bármilyen oldallal a lehető legjobb eredményt adja. A feltételeknek egy korábban elkészített Google Site weboldal felelt meg a legjobban, amely sok nagy méretű képet tartalmaz és a Google neve garancia a jó minőségű hálózatra is. Ez az oldal körülbelül 10 MB méretű, és betöltéséhez körülbelül 10.000 adatcsomag forgalom szükséges. A dolgozat további részében ennek az oldalnak a betöltési idejét és mérési tapasztalatait fogom részletezni.

Az elkészült keretrendszert természetesen más oldalakkal is teszteltem vizsgálva a további fejlesztések lehetőségét, amire az 5. fejezet végén térek még ki részletesebben.

4.2 Mérések végzése

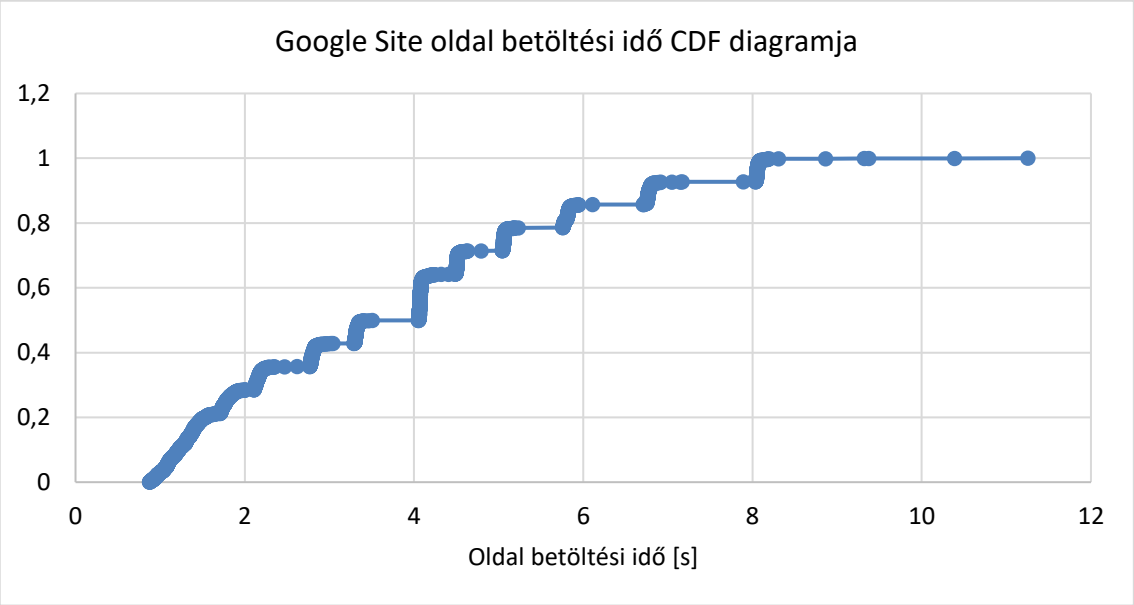
A webböngészéshez a Google Chrome legfrissebb 53-as verzióját használtam. Összesen 2800 oldal letöltést végeztem, amelyhez elmentettem a szükséges HAR és PCAP fájlokat. Különböző felhasználói esetek teszteléséhez a sávszélességet különböző értékekre állítottam 10 és 100 Mbit/s között.

A beállított sávszélesség értékek Mbit/s-ben a következők voltak:

10	12	14	16	18	20	23
26	30	40	50	65	80	100

4-1. táblázat Mérés során használt sávszélesség értékek [MBit/s]

Minden sávszélességgel 200 mérést végeztem így jött össze a (14*200=) 2800 oldalletöltés. A 4-1. ábrán a sávszélesség változás miatti oldal betöltési idő eloszlását láthatjuk.



4-1. ábra Google Site oldal betöltési idő CDF diagramja

A 4-1. ábráról leolvasható, hogy az oldalak betöltési ideje a várakozásoknak megfelelően nagy szórást mutat, de megfigyelhetők csomósodások az adott sebesség melletti átlagos betöltési időknél.

4.3 Szükséges adatok

A gépi tanuló algoritmus betanításához és pontos működéséhez nagyon sok változóra volt szükségem. Ebben az alfejezetben ezeket mutatom be.

4.3.1 HAR fájlok

A Perl-ben írt HAR feldolgozó programommal olvastam ki a pontos oldal betöltési időket. Ez az érték a gépi tanuló algoritmus célváltozója, amelyre feltétlenül szükségem van a betanításhoz és a teszteléshez. Később a betanított algoritmus ezt az értéket fogja megjósolni.

4.3.2 PCAP fájlok

A PCAP fájlokból sokkal több adatot kell kiolvasnom, ugyanis ezek az adatok szolgálnak az gépi tanuló algoritmus bemenetének, ezek alapján történik a tanulás és később a predikció.

A PCAP fájlok feldolgozásának első lépése, a csomagok folyamokra osztása. Ehhez négy paramétert vettem figyelembe, amelyek egyértelműen meghatározzák, hogy melyik csomag melyik folyamhoz tartozik. A figyelembe vett négy paraméter a következő: forrás IP, forrás port, cél IP és cél port. A konkrét mérésben TCP protokollt és SSL csatornát használva böngésztünk, és mivel kizárólag webletöltés történt a használt port szám a 443-as volt.

A már folyamokra osztott csomagokon következő lépésként a szekvencia számok vizsgálatával meghatároztam, hogy hol történt csomagvesztés, hány csomag veszett el és melyik csomagok lettek újra küldve. Ez a szelektálás nagyon mély tudást igényelt a TCP által használt szekvencia számok működéséről, valamint a speciális esetek kezelése is komoly utána járást igényelt.

A csomagvesztés és újra küldés meghatározása utána visszarendeztem a csomagokat az eredeti sorrendjükbe és minden csomaghoz eltároltam az alábbi adatokat:

- a csomag betöltésének relatív ideje (1. csomag a 0. mp)
- az előző csomag betöltése óta eltelt idő
- a csomag mérete
- a csomag iránya – letöltés/feltöltés – 0/1
- az aktuális csomag előtt volt-e csomagvesztés, és ha igen hány darab
- az aktuális csomag korábban elveszett és ezért újraküldött csomag-e – 0/1

Azonban ezek az adatok a konkrét csomagokra vonatkoztak, amelyekből jelen esetben minden PCAP fájlban körülbelül 10.000 darab van. A gépi tanuló algoritmus tanításához és működéséhez azonban nincs szükség ennyi adatra. Így ezekből a változókból leírókat készítettem, amelyek a gépi tanuló algoritmus bemeneteként szolgáltak.

Az alkalmazott 20 leíró adat:

- Két egymást követő csomag letöltési időkülönbségének maximuma
- Letöltési időkülönbségek átlaga
- Letöltési időkülönbségek szórása
- Két egymást követő csomag feltöltési időkülönbségének maximuma
- Feltöltési időkülönbségek átlaga
- Feltöltési időkülönbségek szórása
- Letöltött csomag méretek átlaga
- Letöltött csomag méretek szórása
- Feltöltött csomag méretek átlaga
- Feltöltött csomag méretek szórása
- Letöltött byte-ok száma
- Letöltött csomagok száma
- Feltöltött byte-ok száma
- Feltöltött csomagok száma
- Elveszett csomagok száma
- Hányszor történt csomagvesztés
- Hányadik csomag veszett el először
- Újraküldött csomagok száma
- Utolsó vizsgált csomag időbélyege
- Egymást követő letöltött csomagoknál hányszor történt TTL változás

A fenti 20 leírót és a leírókhoz tartozó oldalbetöltési időt a Perl segítségével fájlba írtam, amely már megfelelő formátumú az gépi tanuló algoritmusok működéséhez.

4.4 Gépi tanuló algoritmus alkalmazása

Az analízis utolsó lépése az előző alfejezetben bemutatott fájl feldolgozása gépi tanuló algoritmus segítségével. A választott gépi tanuló algoritmus a 3. fejezetben részletesen ismertetett Random Forest Regressor. Az összes adatra vonatkozó teljes eredmény kiszámításához azonban még szükség van pár lépésre.

Ahhoz, hogy a teljes adatsorhoz előálljanak a prediktált oldal betöltési értékek, keresztvalidációt kell alkalmazni. Ez az jelenti, hogy az adatsorokat k felé kell osztani. A gépi tanuló algoritmust először $k-1$ osztályon tanítom, amely alapján egy osztályra végig futtatja az algoritmus a predikciót. Ezt a lépést $k-szor$ kell megtenni, hogy minden adaton egyszer elvégződjön a predikció és a kiértékelés. A k általános értékei 3-10. Ezen értéktartományt végig próbálva az eredmények minimális szórást mutatnak, úgyhogy egy középértéket választottam és k értékét 5-nek állítottam be.

4-2. ábra Kereszt validáció

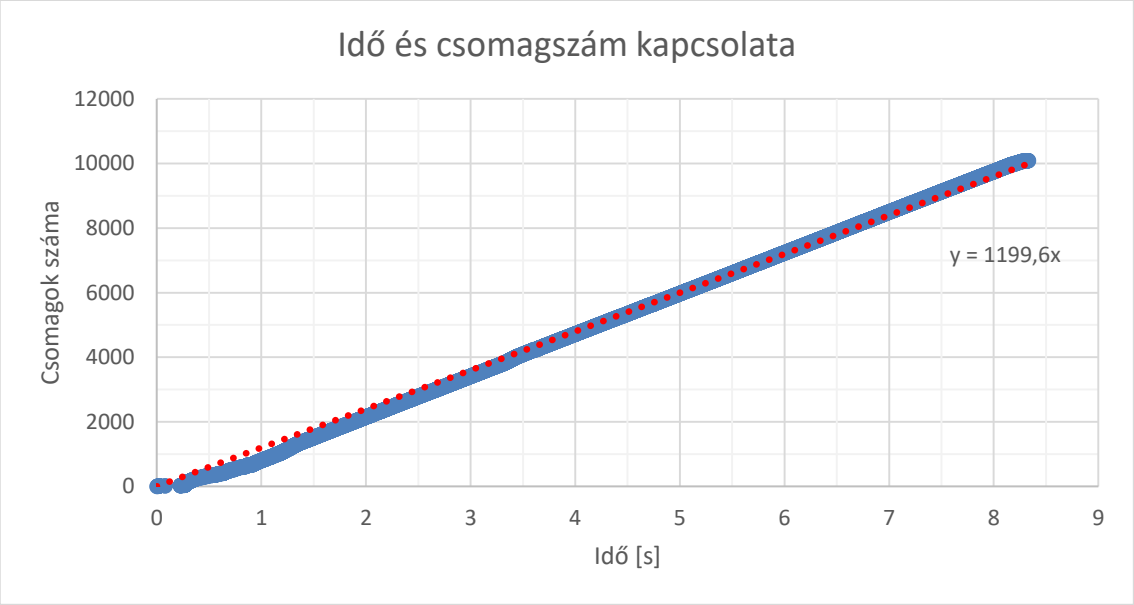
4.5 Vizsgált csomagok számának meghatározása

Az implementáció következő kérdése a vizsgált csomagok számának meghatározása volt. A modell szempontjából ez az egyik legfontosabb paraméter, amelynek meghatározása körültekintést igényel.

Jelen példánál körülbelül 10.000 csomagunk van és a 10.000 csomag információ alapján képeztem a 4.3 fejezetben bemutatott 20 leírót. Ilyenkor nagyon pontosan prediktál az algoritmus. Azonban végig gondolva a folyamatot ez természetes, és elvárható, hiszen ha minden információ a rendelkezésünkre áll, könnyen tudunk következtetni. Így azonban a modellünknek korlátozott felhasználási lehetősége volna, ezért a vizsgált csomagszámot csökkenteni kellett. Ezzel a csökkentéssel duplán nyerünk, egyik oldalról mivel kevesebb csomagot kell megvárnunk, így az oldal betöltéséhez képest sokkal hamarabb, rendelkezésünkre áll az algoritmus futtatásához szükséges adatmennyiség. Másik oldalról az algoritmus futás idején is időt nyerünk,

ugyanis kevesebb csomagra sokkal egyszerűbb bizonyos leíró változók számítása, mint például a szórás jellegű változók esetén. Ugyanakkor a vizsgált csomagszámot nem csökkenthetjük bármeddig, mert az algoritmus pontossága és a vizsgált csomagok száma között egyenes arányosság van.

A csomagszám – pontosság döntés meghozatala előtt még meg kell vizsgálnunk a csomagok időbeli eloszlását. A mérések alapján tudjuk, hogy a kevesebb csomag miatti kevesebb szükséges számítás csak minimális időnyereséget jelent, így a valódi időnyereséget a korábbi vágással, azaz kevesebb csomag vizsgálatával nyerhetünk. A csomagok időbeli eloszlásának vizsgálatához nézzük a 4-3. ábrát.



4-3. ábra Idő és csomagszám kapcsolata

A 4-3. ábráról leolvasható, hogy a letöltés első pár tized másodpercén kívül az eltelt idő és a fogadott csomagszám között lineáris egyenesarányosság van. Az első pár tized másodpercben a TCP protokoll felfutási ideje miatt nincs kihasználva a rendelkezésre álló sávszélesség. Így az ábra alapján a csomagszám választáshoz nincsenek kitüntetett pontjaink, a csomagok nem burst-szerűen hanem folyamatosan érkeztek a várakozásoknak megfelelően.

Az algoritmus predikciója és a valóságos értékek közötti kapcsolat bemutatásához két hiba értéket használtam. Ezeknek a hibaértékeknek a segítségével állapítható meg az algoritmusunk jóság foka. Az egyik az **abszolút százalékos hiba átlaga (Mean Absolute Percentage Error - MeanAPE)**, amely a tesztben szereplő mind a 2800 esetben összehasonlítja a valóság és a predikció közötti hiba százalékot és ezeket átlagolja. A másik az **abszolút százalékos hiba mediánja (Median Absolute Percentage Error - MedianAPE)**. Azért használok százalékos hiba leírókat, mert így könnyebben összehasonlítható a különböző méretű és betöltési idejű weboldalakra történő predikció.

4.6 Idő és erőforrás kérdése

A feladat végrehajtásának egyik legkritikusabb kérdése, hogy mennyi idő és erőforrás szükséges a predikcióhoz. Miért fontos ez a kérdés?

Elképzelésink szerint az elkészült keretrendszerrel minőség becslést lehet adni a böngészési élményre a szolgáltatói oldalon (QoE). De ha kellően gyorsan el tudjuk végezni a predikciót, akár még az oldal betöltése alatt is tudunk változtatni a minőségi paramétereken. Akár mobil akár vezetékes esetben ha látjuk, hogy az első pár csomagnál rossz minőségű volt a kapcsolat, akkor az oldal letöltés további részében nagyobb prioritást adva az adott ügyfélnek a böngészési élmény javítható, illetve összességében nem romlik el. Ezért az esetleges letöltés közbeni közbelépésért fontos, hogy a hiba minél hamarabb minél pontosabban detektálható legyen.

Azonban egy szolgáltató nagyon nagy adatmennyiséget kezel, így ezen a hatalmas adatmennyiségen való számításhoz az idő mellett fontos, hogy program erőforrás igényét is minimálisan tartsuk.

A következő fejezetben ennek a gyakorlati megvalósulását láthatjuk, hogy miként lehetséges az idő és a számítási erőforrások minimális felhasználásával a legpontosabb eredményt elérni

5. Eredmények

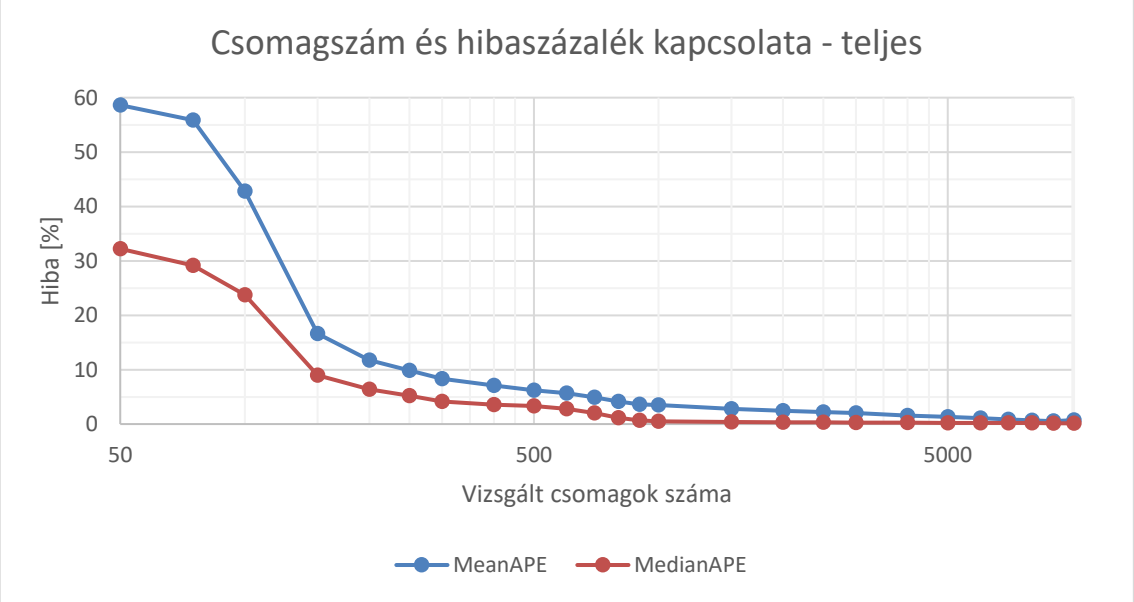
A keretrendszer által generált kimeneteket Excelbe importáltam, és az ott létrehozott szemléletes diagramokon keresztül mutatom be a munkám során elért eredményeket.

5.1 Teljes futási idő összetevői

A 4.6-os Fejezetben részletesen olvashatunk az idő és erőforrás felhasználás minimalizálásának fontosságáról. Ebben a fejezetben a futási idő minimalizálására való törekvés lépéseiről olvashatunk.

5.1.1 Csomagszám meghatározása

A csomagszám a keretrendszer pontosságát a legjobban befolyásoló tényező, amelynek meghatározására méréseket végeztünk. A méréshez a gépi tanuló algoritmust 100 fára és 10-es fa mélységre paramétereztem fel. Ezek átlagos paraméter értékek Random Forest algoritmus használatakor. Azonban végeztem más paraméterek mellett is méréseket, de a függvény átlagos menetére nem volt hatással. A fák számának és mélységüknek az optimumát a következő alfejezetekben bemutatott módon kerestem meg.

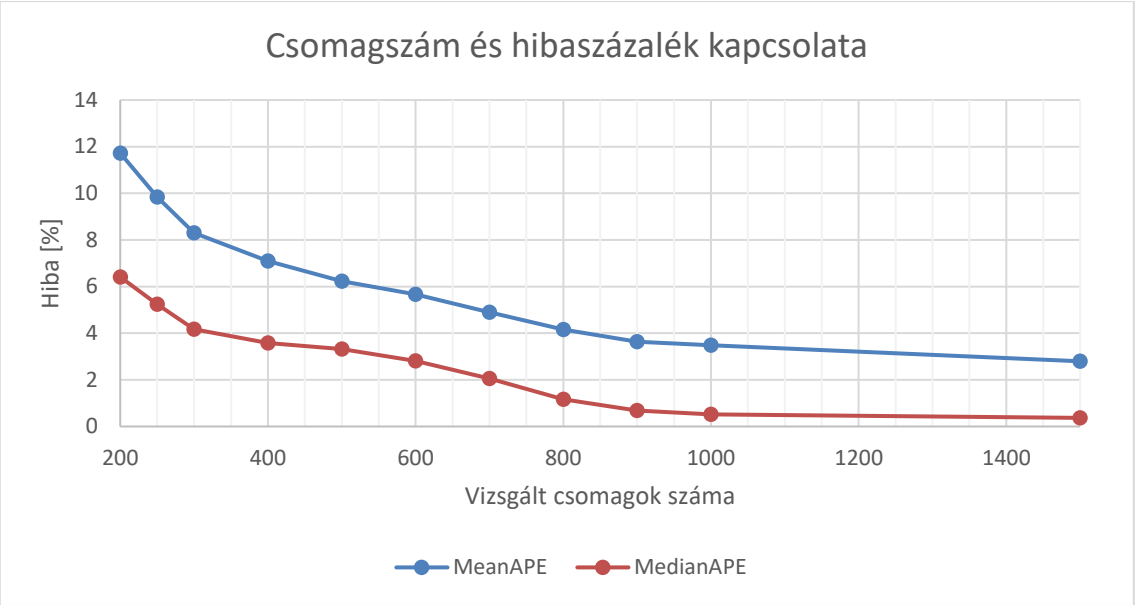


5-1. ábra Csomagszám és hibaszázalék kapcsolata - teljes

Az 5-1. ábráról leolvashatjuk, hogy a modellünk a várakozásoknak megfelelően működik, ugyanis ahogy egyre több csomagot vizsgálunk, csökken a predikció hiba százaléka, míg végül 1% alá csökken.

A másik érdekesség, amit leolvashatunk az ábráról az ábrázolt két függvény különbsége. A két érték között azért van szemmel látható eltérés, mert az átlagos pontosságot pár nagyon rosszul prediktált érték rontja el. Ezt a pár esetet külön megvizsgáltam, hogy miért követ el az algoritmus nagyobb hibákat. Ezeknél az értékeknél a HAR fájlokból kiolvasott betöltési idők jelentősen különböznek az adott sebesség mellett elvárhatótól. Ezt a sebesség különbséget azonban sem a csomagvesztés, sem más hálózati paraméterek nem indokolják, ezért történt a rossz predikció. A HAR fájlt további vizsgálatával arra a következtetésre jutottam, hogy ezekben az esetekben az oldal betöltés a predikciós időhöz közel szinte befejeződött, azonban pár csomagra várni kellett, és ez a várakozás hosszabbította meg az oldal betöltési időt. Ezeket azonban a modellünk segítségével nem tudjuk detektálni, ami nem komoly baj, tekintve, hogy ilyen eset 2800 mérésnél csupán párszor fordult elő és lehet felhasználói szemmel nézve közelebb van a valós betöltési időhöz a prediktált érték, mint a valóságban betöltési időként elmentett időérték.

Az összes csomag vizsgálatánál az átlagos hiba 0,744%, míg a medián hiba 0,140%. Kisebb csomagszámok esetén pedig az átlagos hiba 20% felett van, így a 200 alatti értékeket a továbbiakban nem vizsgáljuk. Azonban nem vizsgálhatjuk az összes csomagot, hanem a korlátozott idő és erőforrások miatt minél kevesebb csomagszám melletti minimális hiba a cél. Így az 5-1. ábra egy szeletét tovább nagyítom a döntéshez. Az ábra alapján a keresett tartomány 20 és 1500 csomag között határoztam meg.

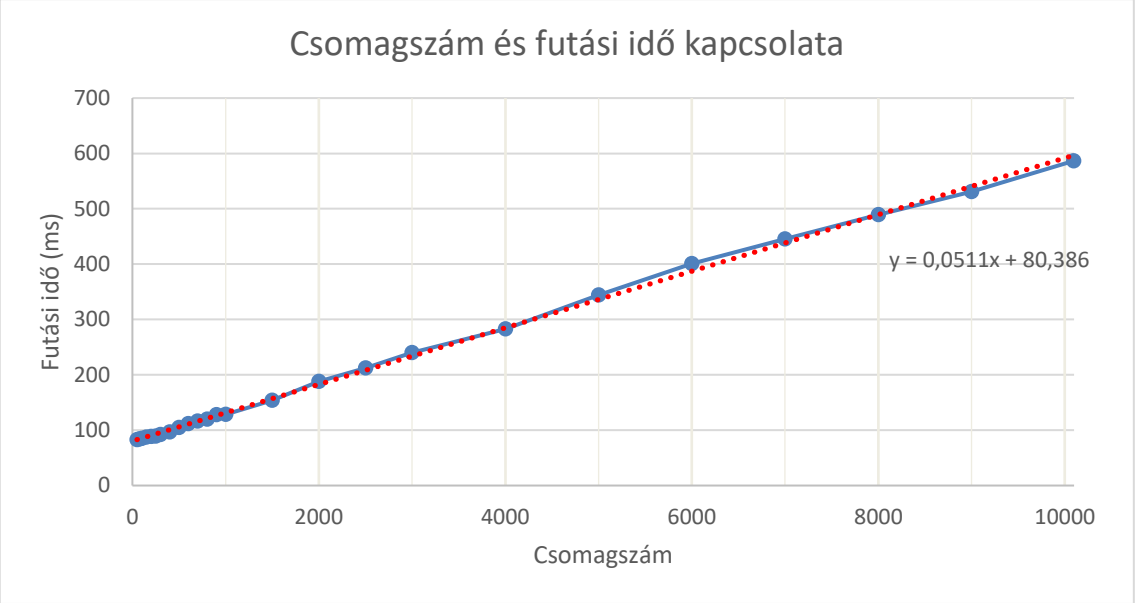


5-2. ábra Csomagszám és hibaszázalék kapcsolata - részlet

Az 5-2. ábra alapján láthatjuk, hogy a vizsgált hiba százalékok csökkenése 900 vizsgált csomagig meredeken csökken utána ellaposodik a függvény. A 900 csomag melletti konkrét hiba értékek 3,7% átlagos hiba és 0,7% medián hiba. A csomagok körülbelül 9%-nak vizsgálata mellett ez egy elfogadható hiba, amely megfelel a további számításokhoz.

5.1.2 PCAP feldolgozó algoritmus futás ideje

A teljes predikcióhoz szükséges idő másik összetevője a PCAP fájlok feldolgozásához szükséges idő. A következő ábrán ezt fogom ábrázolni a vizsgált csomagok függvényében.



5-3. ábra Csomagszám és futási idő kapcsolata

Az 5-3. ábráról leolvashatjuk, hogy a vizsgált csomagszámok és a Perl program futási ideje között lineáris egyenes arányosság van. A korábban kiválasztott 900 csomaghoz 125ms futási idő tartozik. Az egész folyamatot egyben vizsgálva ez a futási idő elfogadható, erősebb feldolgozó géppel tovább javítható.

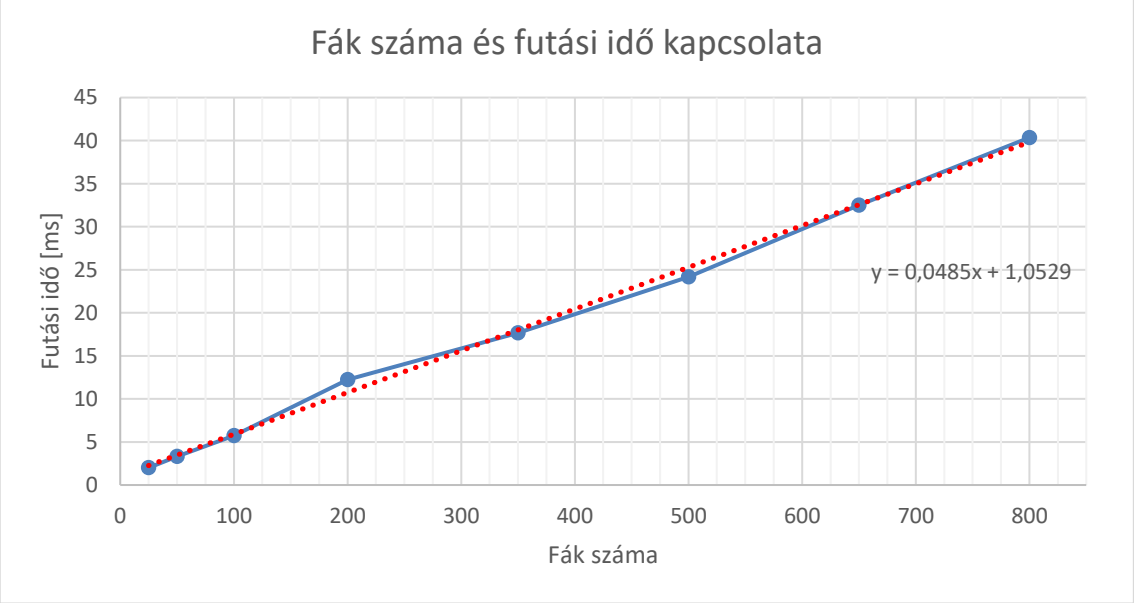
5.1.3 Fák számának és maximális mélységüknek időfüggése

A korábban már kiválasztott és bemutatott Random Forest algoritmusnak két nagyon fontos szintén már bemutatott paramétere van, a fák száma és a fák maximális mélysége. Ennek a két változónak a meghatározására nincsenek konkrét képletek, csak iránymutatások, így a legjobb értékeket tapasztalati úton mérések segítségével állapítottam meg.

5.1.3.1 Fák számának időfüggése

A 4.6-os fejezetben már részletesen bemutattam, hogy az idő és az erőforrás miért nagyon kritikus tényező a feladat megoldásánál. Ennél a döntésnél is egy kompromisszumot kell hoznunk az idő és a pontosság között. Azonban fontos megértenünk, hogy a fák számának változása hogyan befolyásolja az algoritmus futási idejét.

Az 3.6.2-es alfejezetben bemutattam az alkalmazott gépi tanuló algoritmus működését, ahol leírtam a fák szerepét a döntési folyamatban. Mivel minden döntési helyzetben minden fán végig kell futtatnunk az algoritmust, így a fák számának növelésével emelkedik az algoritmus futási ideje és erőforrás igénye. Ezt a függést az alábbi 5-4. ábrán mutatom be.

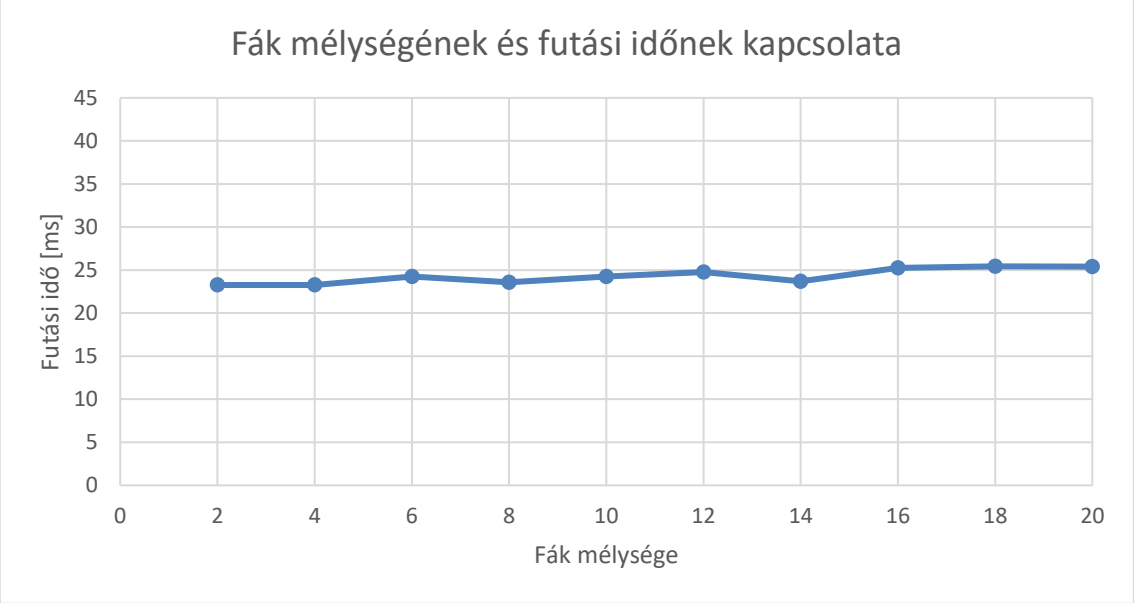


5-4. ábra Fák száma és futási idő kapcsolata

Az 5-4. ábráról leolvashatjuk, hogy a fák száma és a rajtuk futtatott algoritmus futási ideje és erőforrás igénye között egyen arányosság van. Ezt az információt későbbi döntésünk során fel fogjuk még használni.

5.1.3.2 Fák mélységének időfüggése

A következő vizsgálatom a fák mélységének időfüggése. A vizsgálat során a fák számát 500-nak állítottam be, de végeztem teszteket más fa számokkal is, de a mérések szerint a mélység-idő kapcsolat grafikon menete független a fák számától.

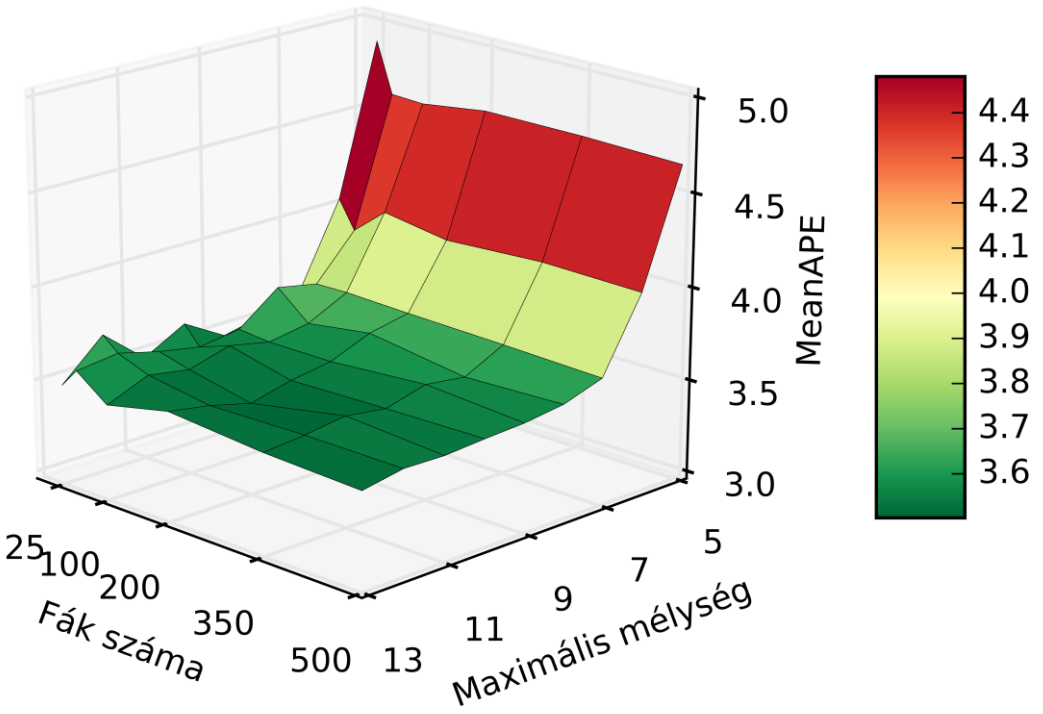


5-5. ábra Fák mélységének és futási időnek kapcsolata

Az 5-5. ábrán láthatjuk, hogy a fák mélysége és az algoritmus futási ideje között nincs kapcsolat, vagyis az algoritmus futási ideje kizárólag a fák számától függ.

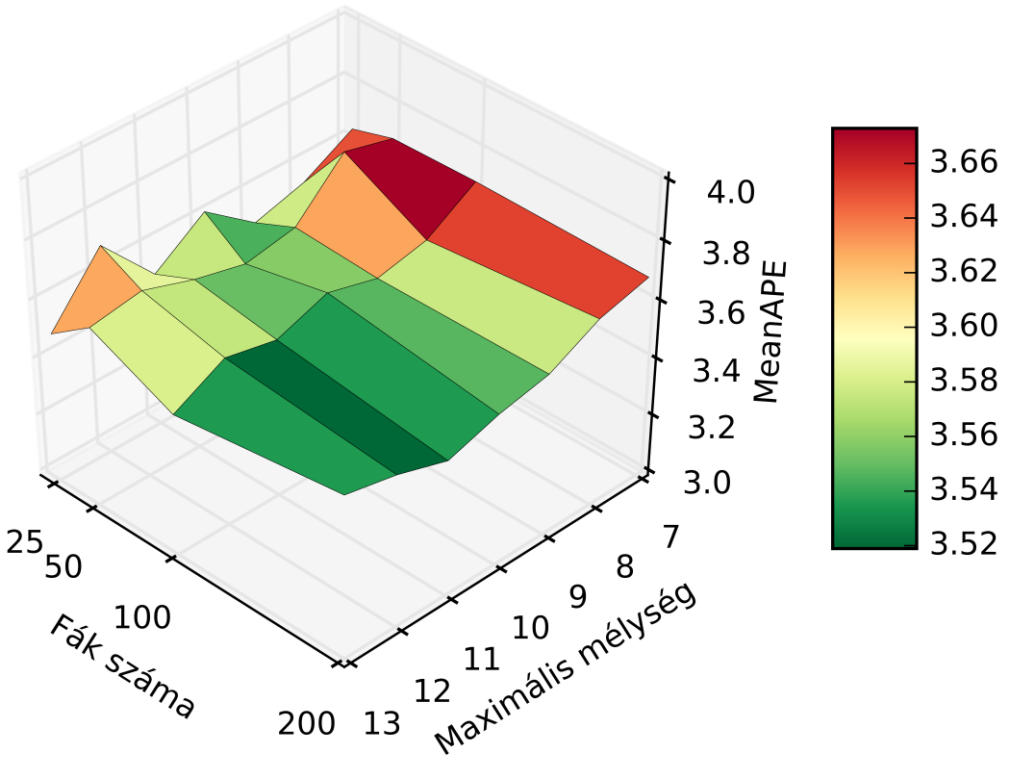
5.2 Fák számának és fák mélységének optimális értéke

A fák számának és a fák mélységének optimális értékének meghatározásához 3D-s ábrát készítettem, amely segítségével vizuálisan dönthetünk.



5-6. ábra Fák számának és maximális mélységüknek hatása a medián hibára

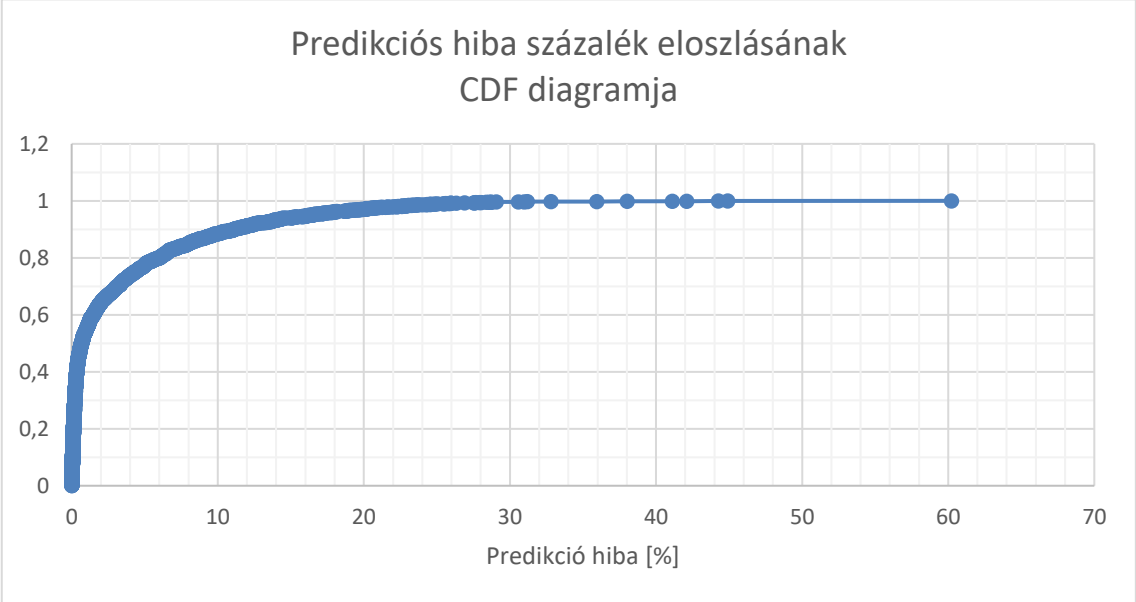
Az 5-6. ábráról leolvashatjuk, hogy 200 fa felett a fák számának növelése már nem jár jelentős javulással, így az erőforrások korlátosságának figyelembe vételével a 25-200 fa 7-12 mélység tartományban keresem tovább az optimális megoldást, amely tartományt a következő képen felnagyítva is megvizsgálhatunk.



5-7. ábra Fák számának és maximális mélységüknek hatása a medián hibára

Az 5-7. ábra alapján a fák számát 200-ra és a fák maximális mélységét 11-re állítva kaphatjuk meg a minimális hiba százalékot, amely 3,558% átlagos hibát és 0,662% medián hibát jelent.

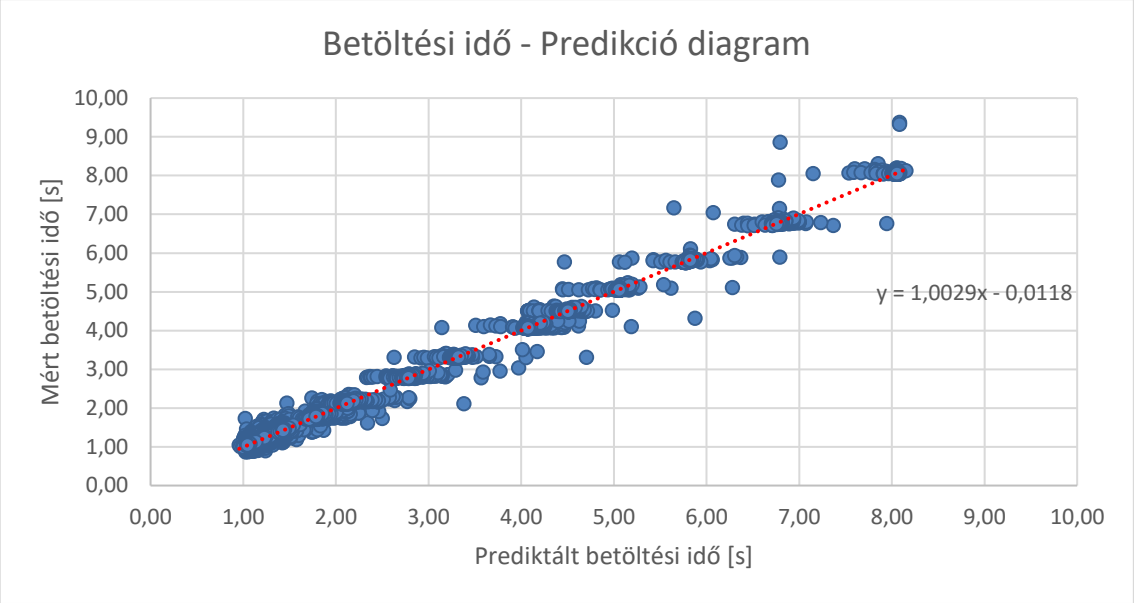
Az 5-8. ábrán ennek a hibának az eloszlását mutatom be.



5-8. ábra Predikciós hiba százalék eloszlásának CDF diagramja

Az 5-8. ábráról leolvasható, hogy az esetek több mint 90%-ban a predikciós hiba 10% alatti. Azonban az 5.1.1 alfejezetben már részletezett okok miatt az esetek kevesebb mint 0,5%-ban 30% feletti hibák is előfordulnak.

Az 5-9. ábrán az összes oldal betöltés valós és prediktált időeredményét láthatjuk.



5-9. ábra Mért és prediktált betöltési idő kapcsolata

A piros színnel feltüntetett trendvonal képlete is olvasható az 5-9. ábrán, amelynek 1,0011-es meredeksége mutatja, hogy a predikció valóban pontos, a trendvonal meredeksége közel van az ideális 1-hez.

5.3 Eredmények összegzése

A beállított paraméterek és a paraméterekhez tartozó adatokat a következő táblázatban foglalom össze 1 oldal betöltésre vonatkozóan.

Sávszélesség	**nagy**	**közepes**	**kicsi**
Oldal betöltés	~2000ms	~5000ms	~8000ms
900 csomag betöltési ideje	~200 ms	~500 ms	~800 ms
PCAP feldolgozó program futási ideje	~ 125 ms		
Gépi tanuló algoritmus futási ideje 200 fa – 11 mélység esetén	~ 12 ms		
Mérőrendszer teljes futási ideje	337	637	937
Fennmaradó idő az oldal betöltéséig	1663	4363	7063
Fennmaradó idő a teljes oldal betöltéshez viszonyítva	83,2%	87,3%	88,3%

5-1. táblázat Keretrendszer futási ideje

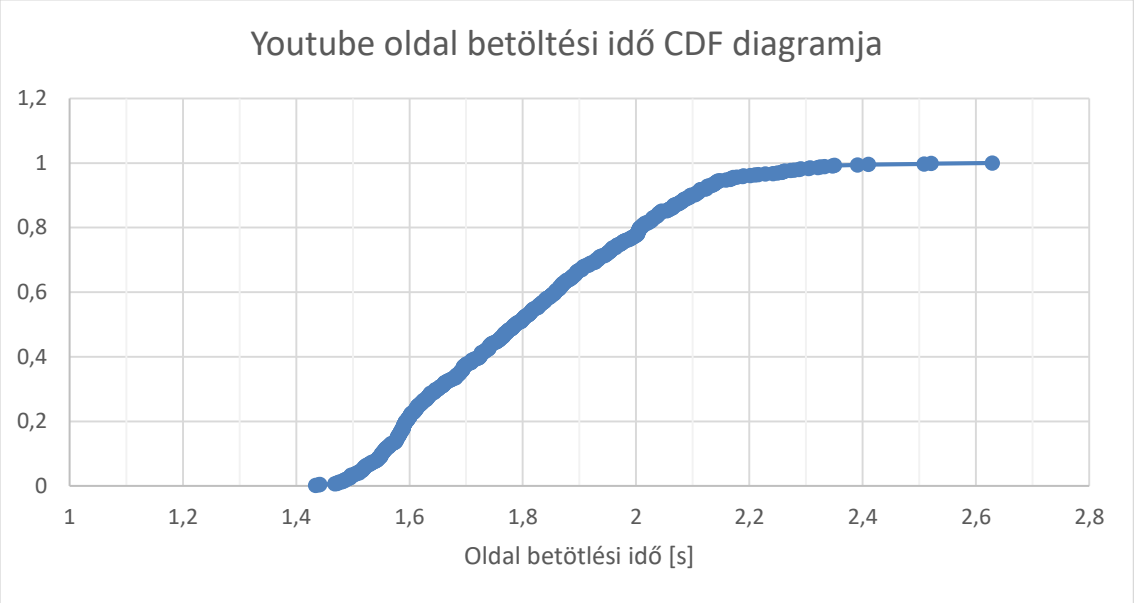
Az 5-1. táblázat utolsó sora alapján kijelenthetjük, hogy sikerült egy elég gyors rendszert létrehoznom, amely segítségével nem csak mérni tudjuk a felhasználói élményt, hanem elegendő időnk van a hálózati paraméterek állításával az esetleges hálózati hibákat javítani a felhasználó tudta nélkül.

Ezen hibaértékekkel teljesítettük célkitűzéseinket, és kellően pontos rendszer jött létre, amellyel szolgáltatói oldalon lehetőség nyílik böngészési élmény becslésére.

5.4 Tovább fejlesztés lehetősége - Más oldalak vizsgálata

A rendszer működésének következő ellenőrzési pontja egy dinamikusabb, kisebb méretű weboldalon történő tesztelés. Ezzel a célunk a valóságos felhasználási felület közelítése, így a legnépszerűbb magyar weboldalak listájáról választottam ki a következő tesztoldalt [23]. A két legnépszerűbb magyar weboldal a Google.com és a Google.hu. Ezek a weboldalak azonban túl statikusak, felépítésük nem hasonlít egy átlagos weboldalra, így bár mérésükkel jó eredményt értünk el, mégsem ezeket hanem a 3. legnépszerűbb oldalt a youtube.com –on történő méréseket mutatom be.

Először is vizsgáljuk meg a youtube weboldal betöltési idejének eloszlását különböző sebességek mellett.



5-10. ábra Youtube oldal betöltési idő CDF diagramja

Az 5-10. ábráról leolvashatjuk, hogy az oldal betöltés idő 1,5 és 2,5 másodperc között mozog. Ez a tartomány sokkal szűkebb, mint a korábban bemutatott esetben. Ez a kevesebb mint 2,5 MB-os oldalméret miatt lehetséges, ugyanis ilyen esetekben a TCP felfutási ideje miatt a nagyobb sávszélességek előnye kevésbé érvényesül.

Az átlagos csomagszám 2700 csomag oldalanként. A korábban kiválasztott a csomagok első 9%-nak vizsgálata mellett maradva, jelen esetben 250 csomagot vizsgálva a következő eredményeket kaptam. A vizsgálathoz 800 oldal betöltést végeztem. Az átlagos hiba 7,567% míg a medián hiba 5,990%.

Ezek az értékek bár magasabbak a korábban mért értékeknél, még mindig 10% alattiak. Konkrét példában az átlagos oldal betöltési idő 1,8 másodperc. Ezzel az értékkel tovább számolva az átlagos hiba 145ms. A számolás alapján látszik, hogy kisebb oldalaknál nehezebb a százalékban kifejezett hibát alacsonyan tartani, hiába a ms-ban kifejezett kicsi hiba.

A mérés tanulsága szerint az implementált keretrendszer, amely a referencia oldalon minimális hibával működik, valós körülmények között is elfogadható hiba százalékkal dolgozik. A mérések számát tovább növelve a hiba tovább csökkenthető és véleményem szerint kellően sok méréssel és a keretrendszer további finomhangolásával bármilyen weboldalt böngészve az átlagos hiba elfogadható szinten tartható.

6. Összefoglalás

A TDK dolgozat készítése során mélyebb ismeretekre tettem szert, az oldal betöltési időket befolyásoló paraméterekkel kapcsolatban, valamint részleteiben megismertem a TCP protokoll felépítését és működését. A kapcsolódó tudományos kutatások elemzésével bemutattam, hogy a mai online világunkban mennyire fontos a hálózati forgalom elemzése, mérése és modellezése, és az így megszerzett tudás hogyan segítheti a jövőbeli hálózatok fejlesztését.

A dolgozatban bemutattam az elkészült keretrendszert fejlesztésének folyamatát és a fejlesztéshez felhasznált eszközöket. Megmutattam, hogy a keretrendszer használatával az oldal betöltési idő kevés csomag vizsgálata mellett is jól prediktálható. Ez a tesztet nem csak a kiválasztott teszt weboldalra, hanem az egyik legtöbbet látogatott weboldalra is elvégeztem.

A mérések további folytatásával felépíthető lenne egy böngészési QoE-t mérő rendszer. Ennek a kiépítéséhez a legtöbbet látogatott magyar weboldalakat kellene végig mérni különböző sávszélességek használatával hasonlóan, ahogy a dolgozatban bemutattam. A pontosság javítása érdekében minden weboldalt meg kellene vizsgálni, hogy hány alapjaiban különböző nézete van. Például egy híroldal mérésénél 2 különböző mérésre lenne szükségünk. Az egyik esetben a híroldal címlapját, másik esetben egy konkrét cikkhez tartozó weboldal betöltése során létrejövő adatforgalmat kellene vizsgálnunk. Véleményem szerint ezzel a két esettel lefedhető lenne egy híroldal. Az esetekre külön-külön tanítanánk be a gépi tanuló algoritmusokat és a böngészés kezdetén az oldallekérést tartalmazó csomag vizsgálatával döntenénk el, hogy melyik előre betanított gépi tanuló algoritmust kell használnunk az adott böngészés élménybecsléséhez. Azonban az adatbázis sosem lehet teljes, de törekedni kell rá, hogy a legtöbb weboldalt lefedje.

A legnépszerűbb oldalakra a vázolt módon betanított algoritmus már széles körben használható lenne, ugyanis a webböngészés során az oldallekérések jelentős része a pár legnépszerűbb oldalhoz tartozik. Ezt kihasználva a szolgáltatóknak lehetőségük lenne, a böngészés közbeni felhasználói élmény becslésére, sőt elegendő idejük lenne akár arra is, hogy az adott felhasználónak nagyobb prioritást adva a rossz felhasználói élmény elkerülhető legyen.

Irodalomjegyzék

[1] The Scientific Association for Infocommunications: Telecommunication Networks and Informatics Services, Hungarian version: Távközlő hálózatok és informatikai szolgáltatások (Contributions: Chapter 1.7 Teletraffic Theory, Hungarian Version: Forgalomelmélet), 2002

[2] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall: Demystifying Page Load Performance with WProf, 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13), 2013

[3] Noriaki Kamiyama, Yuusuke Nakano, Kohei Shiomoto, Go Hasegawa, Masayuki Murata, and Hideo Miyahara: Investigating Structure of Modern Web Traffic, IEEE 16th International Conference on High Performance Switching and Routing (HPSR), 2015

[4] Thierry Spetebroot, Salim Afra, Nicol´as Aguilera, Damien Saucez, Chadi Barakat, Inria Sophia Antipolis: From network-level measurements to expected Quality of Experience: the Skype use case, IEEE International Workshop on Measurements & Networking (M&N), 2015.

[5] Vlado Menkovski, Antonio Cuadra Sánchez, Adetola Oredope, Antonio Liotta: Predicting Quality of Experience in Multimedia Streaming, 7th International Conference on Advances in Mobile Computing and Multimedia, 2009.

[6] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, Hui Zhang: Developing a Predictive Model of Quality of Experience for Internet Video, SIGCOMM'13, Hong Kong, China, August 12–16, 2013

[7] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, He Yan: Modeling Web Quality-of-Experience on Cellular Networks, MobiCom'14, Maui, Hawaii, USA, September 7-11, 2014,

[8] Qi Alfred Chen, Haokun Luo, Sanae Rosen, Z. Morley Mao, Karthik Iyer, Jie Hui, Kranthi Sontineni, Kevin Lau: QoE Doctor: Diagnosing Mobile App QoE with Automated UI Control and Cross-layer Analysis, IMC'14, Vancouver, BC, Canada, November 5–7, 2014,

[9] Nigel Williams, Sebastian Zander, and Grenville Armitrage: A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification, ACM SIGCOMM Computer Communication Review Volume 36, Number 5, October 2006

[10] Speed awareness month: What is a HAR File and what do I use it for? (2016.október) http://www.speedawarenessmonth.com/what-is-a-har-file-and-what-do-i-use-it-for/

[11] HAR Viewer (2016.október) http://www.softwareishard.com/har/viewer/

[12] What is PCAP? (2016.október)
http://www.tech-faq.com/pcap.html

[13] Windump (2016.október)
http://www.winpcap.org/windump/

[14] Wireshark (2016.október)
https://www.wireshark.org/

[15] AutoIt (2016.október)
https://www.autoitscript.com/site/autoit/

[16] CPAN (2016.október)
http://www.cpan.org

[17] CPAN: Archive Har (2016.október)
http://search.cpan.org/~ddick/Archive-Har-0.05/

[18] CPAN: Net::TcpDumpLog (2016.október)
http://search.cpan.org/~bdgregg/Net-TcpDumpLog-0.11/TcpDumpLog.pm

[19] CPAN: NetPacket (2016.október)
http://search.cpan.org/~yanick/NetPacket/

[20] Scikit-learn (2016.október)
http://scikit-learn.org/stable/

[21] Pandas (2016.október)
http://pandas.pydata.org/

[22] Leo Breiman: Random Forest
Statistics Department, University of California, Berkeley, CA 94720, Machine Learning, 45, 5–32, 2001

[23] Alexa Top Sites in Hugary (2016.október)
http://www.alexa.com/topsites/countries/HU

Ábrajegyzék