Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Egyed Tamás Barnabás

KVANTUM SZÁMÍTÁSSAL TOVÁBBFEJLESZTETT KLASZTEREZÉS CENTROID OPTIMALIZÁCIÓVAL ÉS DEEP LEARNING ALGORITMUSOK VIZSGÁLATA AZ ENERGETIKAI SZEKTOR FELADATAIHOZ

KONZULENS

Dr. Ekler Péter

BUDAPEST, 2020

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Egyed Tamás Barnabás**, kijelentem, hogy ezt a TDK dolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens neve) a BME nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig általam a dolgozat feltöltésekor beállított jogosultságokkal (publikus vagy titkos) egyezve közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. A TDK konferenciára való regisztrációval vállalom, hogy az általam publikált anyagot határidőre elkészítem, a konferencián személyesen megjelenek és eredményeimet előadom. A benyújtott tudományos munkát magam (és szerzőtársam vagy szerzőtársaim) készítették, minden, a szakirodalomból átvett bármely rész elérhetőségét – a forrás pontos, más által is elérhető formátumban – szögletes zárójelben megjelölt hivatkozásként megadom.

Kelt: Budapest, 2020. 10. 27.

.....Egyed Tamás Barnabás.....

Egyed Tamás Barnabás

Összefoglaló

Az energetikai szektor, azon belül is az elektromosság felhasználása több, mint 100 éves múlttal rendelkezik. Ezzel szemben a megújuló energia alkalmazása csak 60 évvel ezelőtt kezdődött. A háztáji újrahasznosítható energiát használó erőművek kiépítése ennél is később, pusztán az elmúlt másfél évtizedben indult meg. A zöld energia számos előnnyel bír, ugyanakkor a jelenleg működő hálózati infrastruktúra nem feltétlenül képes az új kihívásokra a megfelelő módon reagálni – több veszteséges tényező is felmerül az aktuális rendszerben. Új lehetőségeket nyit meg a hálózat fejlesztése, melyben nagy szerepet játszanak a frissen telepített, úgynevezett „okos" mérőórák.

Az energetikai hálózat a következő években el fog jutni arra a szintre, hogy nem csupán nagy területekről lehet majd aktuális fogyasztási adatokat kinyerni, hanem az új mérők révén végfogyasztók esetén is rendelkezésre állnak majd ezek. Természetesen olyan megkötésekre továbbra is számítani kell, mint hogy az – akár negyed óránként mért – információk csak naponta egyszer kerülnek be a központba, hogy ne legyen túl nagy az adatterhelése az új hálózatnak. Ez a fajta monitorozás viszont így is elégséges információt biztosít, hogy gépi tanulást felhasználva a működés optimálisabb legyen.

A kutatás célja, hogy meghatározásra kerüljön mely mesterséges intelligencia és adatkezelési algoritmusok alkalmasak a hatékonyabb működés elérésére. A felhasználókat az ilyen jellegű alkalmazások esetén többek között csoportosítani is szükséges. Egy esetet kiemelve például, ha a halmazok kialakítását a számítógépre bízzuk, úgy lehetőség nyílik illegális áramfogyasztásokat is észlelni. Másik nagyobb gondolatkör a predikció. A megújuló energiaforrások kiszámíthatatlansága nagy problémát jelent az ipari és otthoni erőművek termelési kapacitásának beállításakor, melynek során a hálózatban energiatöbblet alakulhat ki. Ezt oldja meg a kutatás alatt készített algoritmus és megoldás, mely időjárás előrejelzés alapján képes megjósolni a háztáji napenergia termelés alakulását.

A dolgozatban ismertetésre került két klaszterező-, valamint három predikciós algoritmus. A meglévő algoritmusokra alapozva újakat terveztem és valósítottam meg, melyek megoldást nyújtanak az adott problémára. Ezen eszközök összehasonlítása során, a mérésekből és az azok alapján készült diagramokból ki lett választva, mely megoldás a legjobb az adott feladatra. A kijelölt klaszterező metodika hatékonyság javító technikái kidolgozásra kerültek, úgy mint a centroid optimalizáció, hierarchikus klaszterezés, továbbá a kvantum számítás – mint a következő évtizedek számítási sebességének meghatározó tényezője. Az ismertetett fejlesztés a mérések alapján jó eredményeket ért el.

Abstract

The energy industry within that the consumption of the electricity has more than 100 years long history. In contrast the use of the renewable energy started only in the last 60 years. The construction of power plants using renewable energy in the consumers home started even later, only in the last decade and a half. The green energy has many advantages however the actually working network infrastructure is not able to react in the appropriate way on the new challenges – there are several loss factors in the system. The development of the network with "smart" meters opens up new opportunities.

The energy industry's network will step on a new level in the next few years. The system going to be able to present the actual energy consumption data of the end-users. The intelligent meters could monitor the energy usage in every 15 minutes. Of course, there will be some constraints like the information about the consumption will be delivered one time per day by the way the data load on the new generation network will be too high. This type of monitoring gives fairly enough data to build some deep learning solutions to optimize the system.

The purpose of the research was to investigate some artificial intelligence algorithms which can be applicable to increase the efficiency of the network. The consumers have to be classified. Only one of the use-cases is that the computer is able to make clusters from the consumers and with this type of information it can recognise those who stole the energy from the system. Furthermore, there are approaches of the prediction. A great problem of the renewable energy sources is the unpredictability which causes excess energy in the network. That is what some of the programs solves which were created during the research. The solutions use current weather forecasts and energy production data from the past.

The study explained two clustering- and three prediction algorithms. From the existing algorithms, new algorithms have been developed that provide solution to the given problems. The best solutions were selected in terms of the given aspect of use. The comparison was made based on the research measurements and the diagrams of these. Techniques to improve the efficiency of the selected clustering methodology have been developed. Such as centroid optimalization, hierarchical clustering and quantum computation – one of the main factors of the computing speed in the next decades. The described solution achieved good results based on the measurements.

1 Bevezetés

A gépi tanulás, mint elnevezés 1959-ben lett definiálva. Maga az ötlet, hogy a számítógépek egyszer képesek lehetnek maguktól tanulni, még ennél is régebbi. Az 1970-es években egy kutatócsoport mutatott rá a tudás alapú rendszerek legnagyobb hiányosságára [1]. A kérdésre, hogy maga a tudás, az honnan származik? Az alapkoncepció azt mondja ki, hogy területi szakértők kommunikálnak a programokat készítő mérnökökkel, akik megvalósítják az általuk elvárt rendszert. Ez a módszertan gyakran frusztráló, hiszen a két különböző fél más-más „nyelvet" beszél hivatásukból adódóan. Ennek ellenére közösen kell megfogalmazniuk több ezer „ha ..., akkor ..." típusú állítást, ami a rendszer „tudását" alkotja majd. Az alapkoncepció javítására alakult ki az az ötlet, hogy imperatív megfogalmazás helyett, inkább kapjanak indirekt utasításokat a szerkezetek. A szükséges képességeket pedig tanulják meg a gépek példák segítségével.

1983-ban jelent meg az a kutatói anyag (Machine Learning: The AI Approach - Edited by R. Michalski, J. Carbonell, and T. Mitchell.), ami berobbantotta a gépi tanulás elképzelést a szakmai tudatba. Ekkor még nem csak a hogyan volt a kérdés, hanem a mit és miért lenne szükséges tanulnia a gépeknek. Több kutatás irány is elindult, melyek közül nem mind lett sikeres. Tom Mitchell könyve 1997-ben került kiadásra (T. Mitchell, Machine Learning, McGraw-Hill (1997)), mely szintén meghatározónak számít.

Érdemes kiemelni, hogy a gépi tanulás („machine learning") és a mély tanulás („deep learning") összetartozó fogalmak, így ebben a dokumentumban nem kerül részletezésre, hogy melyik algoritmus pontosan melyik témakörbe lenne sorolható. Röviden a következőképpen jellemezhetőek. A gépi tanulás a mesterséges intelligencia megoldások egy olyan alcsoportja, amely strukturált adatokkal dolgozó, a saját hatékonyságának fejlesztéséhez emberi beavatkozást nem igénylő módon éri el a kívánt eredményeket. A mély tanulás lényegében a gépi tanulásra épít. Azt próbálja modellezni, ahogyan az emberi agy működik. Szokás úgy tekinteni rá, mint gépi tanulási algoritmusokból épített hálózatokra, hiszen az elnevezés arra utal, hogy egymásra épülő (rétegzett) – az adatokat eltérő szempontok szerint elemző – lépésekből áll.

Jelen dokumentum kiemelten az energetikai szektorhoz kapcsolódó felhasználási lehetőségeket veszi alapul, s ezekhez kínál megoldási ötleteket, javaslatokat. A következőkben bemutatásra kerül néhány tipikus kutatási és alkalmazási terület.

1.1 Klaszterezés

A csoportosító, vagy idegen szóval klaszterező (cluster) algoritmusokkal érdemes elkezdeni a vizsgálódást, hiszen ezek képesek felügyelt környezet és előre megadott tanulási halmazok nélkül is csoportosítást végezni a bemenő adatsokaságon. Az így kialakított egységekre további gépi tanulásra alkalmas módszereket lehet építeni.

A klaszterezés egy gyűjtőfogalom. A csoportosítás folyamata úgy kerül végrehajtásra, hogy a bemenetként kapott adathalmazt vektoroknak tekinti, ahol az adott értékek egy-egy attribútumnak az értékei. Az ilyesfajta algoritmusok jellemzően nem veszik figyelembe a szintaxist, csupán az értékekre összpontosítva alakítják ki az összetartozó halmazokat. Kimenetként a létrejött „x" darab klaszter szolgál. Fontos, hogy a csoportok nem lapolódhatnak át, tehát az eredmény diszjunkt halmazokból kell álljon. Az egyik legrelevánsabb kérdés az, hogy hány klaszter fog létrejönni, és erre nem létezik egyértelmű válasz. „Távolról" nézve egy adott ponthalmaz egy csoportot alkot, viszont ahogy egyre „közeledik" a megfigyelő, úgy egyre több összetartozó egységet lehet felfedezni. Fontos, hogy az algoritmusoknak explicit meg kell adni, hogy milyen eltérés számítson még egy halmaznak, vagy éppen azt, hogy mennyi csoportot szeretne kapni az adott felhasználási eset kimenetként. Ez pontosan az imént említett szemléletnek feleltethető meg. Azzal, hogy a kívánt távolság, vagy éppen elvárt csoportszám adott, pontosan azt az információt lehet rögzíteni egy másik formában, hogy milyen „távolról" kerül láttatásra az adatsokaság (ez az emberi nyelven leginkább érthető szemlélet mód, viszont a gépek számára a másik kettő bemeneti paraméter sokkal több információt ad).

A dolgozat célja, hogy több klaszterező megoldás közül kiválasztásra kerüljön saját mérések segítségével, hogy fogyasztási adatok alapján mely algoritmus a leghatékonyabb csoportosító metódus. Ezen metodikák alkalmazhatósága, illetve hatékonyság növelése végett az adott adathalmazhoz illeszkedő előfeldolgozási lépések is meghatározásra kerülnek. A választott algoritmus továbbfejlesztése megtörtént.

1.2 Predikció

Második lépésben érdemes az úgynevezett predikciós algoritmusokra fókuszálni. A tudomány főtémakörei közé tartozik a jövőbeli események megjóslásának gondolata. Ez számos esetben megvalósítható, ugyanakkor akadnak olyan helyzetek, ahol a kapott adatsorozat valóban véletlenszerű, és nem lehet az elkövetkező időszakra következtetéseket adni.

A predikciós algoritmusok felhasználási területe határtalan. Az energetikai szektorhoz kapcsolódóan is számos aspektusban alkalmazhatók. Például az időjárási körülmények, illetőleg a korábbi fogyasztási adatok felhasználásával – ahol a háztáji termelők, akik napelemeket működtetnek, már csoportosításra kerültek egy klaszterező algoritmus által –, viszonylag pontosan megadható, megjósolható, hogy mennyi energiatermelés várható az adott ellátási blokkokban, s ehhez igazítható, hogy mennyi energia eljuttatására lesz még szükség egy adott szegmensben. Ennek segítségével csökkenthető a hálózat felesleges terhelése. Másik láthatóan fontos felhasználási eset (use-case) a predikciós algoritmusok számára az, hogy segítségükkel bizonyos változások alapján automatikus értesítéseket kaphatnak a rendszerkezelők egy esetleges hiba bekövetkezésének valószínűségének növekedéséről. Vagy éppen az éves szinten egyszer, a vevők számár meghatározott havi általány díjak is pontosabban jósolhatók ezen metódusok használatával. Szintén érdekes lehet a felhasználók automatizált értesítése, ha valamilyen új fogyasztó miatt, vagy egyéb okból a meghatározott általány értéktől feltehetően jelentősen eltérő fogyasztások adódnának, ezzel kiküszöbölve az év végi jelentős ráfizetéseket a vásárlók számára, valamint a felesleges jóváírásokat a szolgáltató részéről (ez jelentős bevétel kieséssel járhat az ipari szereplőknek, így viszont egész évben egységes maradhat a pénzügyi rendszer). Utolsó példaként megemlítve a hálózati struktúrából adódóan elképzelhető, hogy egyes mérők adatait a rendszer valamilyen oknál fogva nem tudja rögzíteni, vagy csak nem tudja eljuttatni a központba. Ennek a megoldását is a predikció adhatja, hiszen segítségével kikövetkeztethető, hogy a hiányzó érték mi lehet.

A kutatás motivációja, hogy több predikciós algoritmus közül kiválasztásra kerüljön, jelen körülmények között készített mérések segítségével, hogy időjárási adatok alapján mely algoritmus képes a leghatékonyabban megjósolni a napsugárzás mértékét, s ezzel a napelemek hatásfokát. Ezen metodikák használhatósága, illetve javítása végett az adott adathalmazhoz illeszkedő előfeldolgozási lépések is bemutatásra kerülnek.

1.3 Kvantum számítás

A kvantum számítás velejárója, hogy csak bizonyos aspektusokban használható jól, de azokban ténylegesen nagy teljesítménnyel. Megfelelő célokra, mint a hatványozás, illetve egyéb más matematikai műveletek, melyek a mai normál számítógépek számára nehezen elvégezhetők, kiemelkedő hatékonysággal alkalmazható a kvantumszámítás.

A fentiek alapján elmondható, hogy amennyiben sikerül megtalálni az adott mesterséges intelligenciát alkalmazó megoldásokban azokat a lépéseket, amelyek matematikailag kiválthatók kvantum algoritmusokkal, vagy mindenesetre megfelelő közelítést, becslést lehet megadni segítségükkel, akkor hibrid megoldások képezhetők, melyek bizonyos költséges – értsd erőforrásigényes, teljesítménykritikus – műveleteit ki lehet szervezni. Jelenlegi tudásunk alapján egy teljes AI algoritmus kvantum számítógépre történő implementálására nincs lehetőség, de ezt a problémát kiküszöböli ez a megoldás, hiszen így mégis használhatóvá válnak a kvantum processzorok az adott helyzetben.

A kvantumszámítás alapjaiban különbözik a megszokott számítógépektől. Ebben a környezetben nem értelmezettek a bitek, nem egyesekkel és nullákkal működnek. Ehelyett úgynevezett kvantum biteket („qubit”) lehet alkalmazni, melyek bizonyos időpillanatokban mérések segítségével olyan információt szolgáltathatnak, amit mégis megfeleltethetünk az eddigiekben értelmezett biteknek. Ez úgy történik, hogy a számítások során a qubitek manipulációjával ezekhez egy időben változó valószínűségi érték keletkezik, ami természetesen végtelen sokféle lehet. Érthető módon ritkán lesz az adott qubit tisztán 1-es, vagy 0-s állapotban, ehelyett legtöbbször egyszerre veszi fel a két értéket, vagyis ezek valamilyen szuperpozíciójába kerül. Ugyanakkor a mérések során, adott időpillanatban eldönthető, hogy éppen melyik állapotba áll be a qubit, s így egy tisztán egyes, vagy nullás állapotot reprezentál, amit a jelenlegi számítógépes rendszerek is értelmezni tudnak.

Mivel a kvantumszámítás még jelenleg is fejlődik, illetve nehezen hozzáférhető közhasználatra egy ilyen jellegű fejlesztő eszköz, ezért még nem is lehet magasabb szintű programozási lépésekben, nyelvekben gondolkodni ezen területen. Áramköri szinten viszont lehetőség van szimulálni az ilyesfajta gépek működését, így ezekből már most is építhetők algoritmusok. Hasonlóan a normál informatikához itt is úgynevezett kapukat („gate”) lehet felhasználni az áramkörök építéséhez. Ezeket a kapukat sokkal inkább érdemes úgy elképzelni, mint egy három tengelyű („x”, „y”, „z”) koordinátarendszerben, és az arra felrajzolható egység gömbben, történő forgatásokat. Ezzel a nézettel természetesen adódik, hogy a komplex számok kerüljenek előtérbe, hiszen azokat is lehet ily módon ábrázolni, és ezek számok révén értelmezhető prezentációt nyújtanak a számítógépeknek, továbbá matematikai műveleteket tesznek elérhetővé.

Mindezek alapján elmondható, hogy a jelenlegi kvantum programok úgy működnek, hogy ki kell választani a használandó qubiteket, ezeket kiinduló állapotúra kell inicializálni, majd kapuk sorozatain lehet ezeket végig vezetni. A végeredmény a kiinduló qubitek végső állapotából adódik, de természetesen elképzelhető, hogy valamely egység csak segéd szerepet tölt be az algoritmus során, így a kalkuláció lehet csupán egy qubit adott pillanatban mért értéke is.

A klaszterező algoritmus továbbfejlesztése során egy olyan metódus bevezetése a cél, mely általánosan használható mesterséges intelligencia megoldások kvantumszámítógépen történő végrehajtására – olyan értelemben, hogy az AI metódus jól meghatározott matematikai része/egységei kvantum környezetben végrehajthatóvá válnak.

1.4 Irodalomkutatás

Mind a mesterséges intelligencia terjedése, mind a kvantum számítógépek fejlődése előre vetíti azt, hogy az elmúlt és elkövetkező években gazdag irodalomgyűjtemény született, valamint születik majd. A következő bekezdések egy-egy könyvet ismertetnek, amelyek relevánsak lehetnek az *MI* eszközökkel, vagy éppen *kvantum számítással* dolgozó mérnökök számára.

Miroslav Kubat tollából származik az *An Introduction to Machine Learning* [1] című könyv. Ez az írás matematikai precizitással, háttérrel, illetve ábrákkal támogatva mutatja be a gépi tanulás alapjait, különböző aspektusait. Ezen felül kitér a terület különböző csoportjaira, így például a felügyelt és nem felügyelt tanító környezetekre is, melynek megfelelően részletesen meg lehet ismerkedni a klaszterezés elméletével. Különböző osztályozó algoritmusoktól kezdve, a regresszión át, a genetikus algoritmusokig lehet tudást szerezni ezen mű segítségével.

A *Algorithmic Aspects of Analysis, Prediction, and Control in Science and Engineering* [2], Jaime Nava, valamint Vladik Kreinovich könyve. Ez az alkotás többek között – témába vágóan – a deep learning egyik fő területével a predikcióval, a megjósolhatósággal foglalkozik. Pszeudokódokkal, matematikai levezetésekkel mutatja be a predikcó lehetségességét, számítási módokról értekezik.

Az *Elements of Quantum Computing: History, Theories and Engineering Applications* [3] című kötet eltér az eddig felsoroltaktól, hiszen ez már magával a kvantum számítással foglalkozik. Seiki Akama arról ír, hogy mit is neveznek kvantum

számításnak, milyen számítási modellek léteznek manapság, továbbá ismerteti az alapokat (qubit, kapuk, …). Megemlíti a kvantum számítógépek felhasználási lehetőségeit, illetve egy jövőképet is vizionál a fejlődésükkel kapcsolatban.

Végül de nem utolsó sorban egy másik hasznos írás a *Quantum Computation and Quantum Information* [4], melyet Michael A. Nielsen és Isaac L. Chuang készített. Ez a kifejezetten részletes könyv bemutatja a kvantum informatika alapjait, kitér a távolság becslésre/mérésre kvantum számítások segítségével, valamint függelékében számos érdekes, aktuális felhasználási területet is bemutat.

Jelen munka a terület adta lehetőségeket ismerteti az energetikai szektor aspektusából, több különböző algoritmus alkalmassági, hatékonysági vizsgálata, valamint egy kiválasztott algoritmus továbbfejlesztési lehetőségeinek bemutatásán keresztül.

1.5 Fejezetek ismertetése

A dolgozat az alábbi módon épül fel.

- A 2. fejezetben bemutatásra kerülnek a klaszterező, illetve predikciós algoritmusok.
  - A 2.1-es alfejezet ismerteti a klaszterező algoritmusokat, valamint a hozzájuk kapcsolódó előfeldolgozási lépéseket, mérési eredményeket, majd a döntést, hogy az adott aspektusban melyik metodika bizonyul a legjobb választásnak.
  - A 2.2-es alfejezetben a predikciós algoritmusok, ezek előfeldolgozási lehetőségei, valamint a mérések alapján legjobb metódus kiválasztása szerepel.
- A 3. fejezet a K-Means továbbfejlesztési lehetőségeiről szól.
  - A 3.1-es alfejezet részletezi a fejlesztéshez szükséges adatgenerálás menetét.
  - A 3.2-es alfejezet bemutatja a hierarchikus klaszterező algoritmus futtatás adta lehetőségeket, és ajánlást tesz az efféle végrehajtás időzítésére is.
  - A 3.3-as alfejezet a K-Means algoritmus centroid optimalizációval történő kiegészítést mutatja be és igazolja hatékonysági mérésekkel.
  - A 3.4-es alfejezet ismerteti a javasolt kvantum megoldást, mely képes együttműködni a K-Means és más AI metodikákkal.
- Az 4. fejezetben a továbbfejlesztési lehetőségek és a dolgozathoz kapcsolódó összefoglaló gondolatok szerepelnek.

2 Algoritmusok vizsgálata az energetikai szektor feladataihoz

Ez a fejezet hivatott ismertetni a tanulmányban megvizsgált algoritmusokat, adott – az energetikai szektorhoz kapcsolódó – problémák mentén. Ennek során bemutatásra kerül az algoritmusok felépítése, pszeudokódja, valamint az elkészült prototípusok teljesítmény metrikái is megjelennek, diagrammokkal reprezentálva.

2.1 Klaszterező algoritmusok

Az adatok megfelelő csoportosítása minden problémakörben kiemelten fontos, így nincs ez másként az energetikai szektorral sem. A mesterséges intelligencia algoritmusok egy nagy csoportját jelentik az úgynevezett klaszterező algoritmusok. Ezek az adatok átható ismerete, továbbá egyéb metainformációk nélkül is képesek – úgynevezett felügyelet nélküli környezetben – a céljukat elérni, halmazokat kialakítani. A létrejött csoportok azért is fontosak, mert más felügyelt környezetben működő algoritmusok számára alapot adnak, hiszen az adott adategység hovatartozása egy elégséges előzetes tudásnak tekinthető.

2.1.1 K-Means Clustering

A K-Means jelenleg az egyik legnépszerűbb, irányítatlan klaszterező algoritmus. A módszer lényegében úgy indul, hogy létrehoz „k” darab kiinduló klasztert, úgy, hogy mindegyik bemenetként kapott elem (adatvektor) valamely csoport részét képezze. Ezek után minden egységre meghatározásra kerül egy centroid, azaz az alakzat (ami közre fogja a pontokat – vagyis a bemeneten kapott adatvektorokat) geometriai közepe.

A következő lépésben minden egyes pontra kiszámolja az algoritmus mind a „k” darab centroidtól vett távolságot, és úgy módosítja a csoportokat, hogy az elemek a hozzájuk legközelebb eső középponttal rendelkező klaszterbe kerüljenek. Ezek után azokra a csoportokra, ahol változás történt újra ki kell számítani a centroidot. Erre azért van szüksége, mert amennyiben egy elem el lett távolítva/ hozzá lett adva a halmazhoz, akkor máshova tolódik a középpont. A vektorok normalizálása segíti az algoritmus működését, így érdemes megfontolni azt. Normalizálni a következő matematikai képlet alapján lehet: $x_{norm} = \frac{x - MIN}{MAX - MIN}$.

Az algoritmus futásának kilépési feltétele abból fakad, hogy előbb-utóbb eljön az az állapot, amikor minden pont a hozzá legközelebbi klaszterbe kerül. Ekkor a csoportok diszjunktak, és minden pont hozzá van rendelve egy egységhez, azaz elérte a végrehajtás a célját, kialakult a „k” darab halmaz.

A K-Means algoritmus előnyei közé tartozik, hogy jól érhető és matematikai összefüggések alapján implementálható. Ugyanakkor a legnagyobb gond ezzel a metódussal az, hogy jelentős mértékben függ a kialakított csoportosítás a kiinduló klaszterektől. Ugyanazon adatvektorokon futtatva meglehetősen különböző végeredményekre juthat a módszer, de ez például centroid optimalizációval, vagy hierarchikus klaszterezéssel kiküszöbölhető. Egy másik probléma, hogy a felhasználó nem feltétlen tudja azt, hogy hány csoportot szeretne, mégis meg kell adni a megkövetelt klaszterek számát.

A K-Means exponenciális komplexitású – $O(n^2)$ –, ugyanakkor gyakorlatban gyakran lineáris algoritmusként lehet felhasználni – $O(n)$. Korlátozó tényezőket is be lehet vezetni, ilyen lehet egy megfelelő felső korlát választása az iterációk számára.

A következő szakasz az algoritmus fő elemeit ismerteti.

Bemenet:

- egy adatvektor halmaz (pl. CSV fájl), címkék, attribútum elnevezések nélkül
- „k” konstans

Lépések:

1. „k” db kiinduló klaszter létrehozása, ezekben centroid számítás (Ci)
2. Egy vektor (x) kiválasztása és az összes meghatározott centroidtól vett távolságának a kiszámítása. Ekkor „j” jelölje a leközelebbi centroid indexét.
3. Ha „x” eredetileg is a „j”-edik klaszter eleme volt, akkor nem kell csinálni semmit. Egyébként „x” jelenlegi csoportjából át kell helyezni őt a „j”-edik halmazba és újra kell számolni a lépéshez kapcsolódó két centroidot.
4. Amíg a kilépési feltétel nem teljesül ismételni kell a 2. és 3. lépést.

Kilépési feltétel:

- Minden adatvektor a hozzá legközelebb eső klaszterben van.

Kimenet:

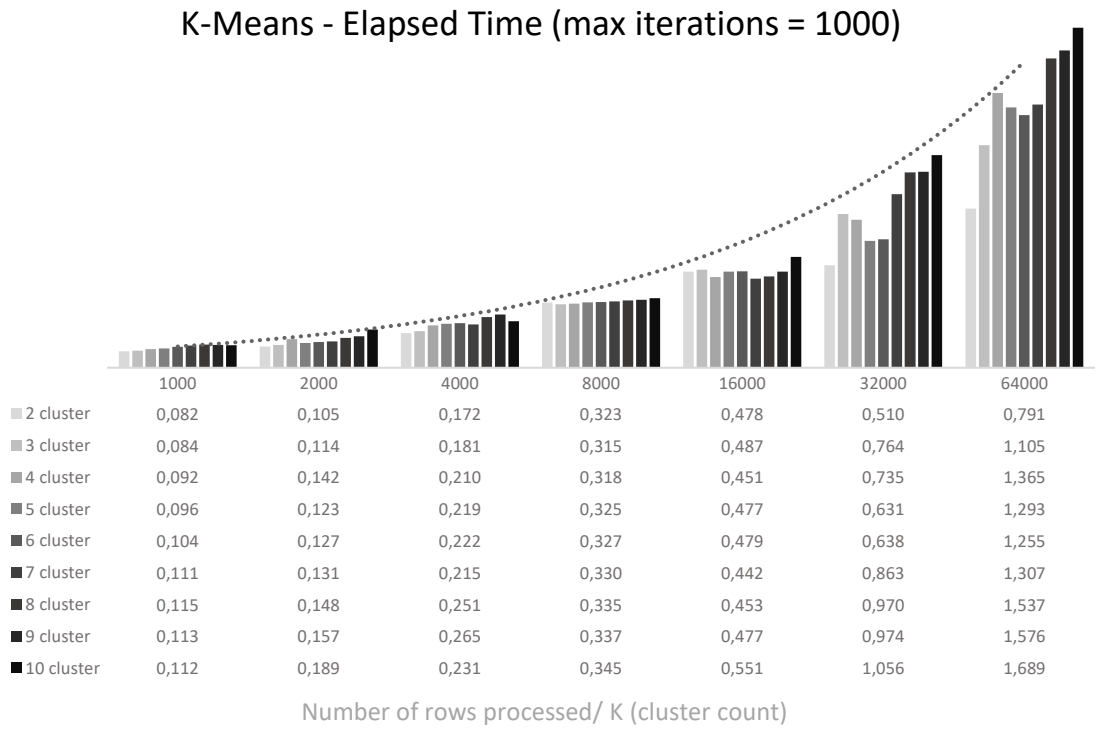
- „k” darab klaszterből álló csoportosítása a bemenetként kapott vektoroknak.

Az alábbiakban a mérési eredmények kerülnek bemutatásra.

A K-Means szemléltetéséhez, illetve a mérésekhez elkészült egy prototípus, melyet a „kmeans-proto” program mutat be. A beolvasás, mint adathalmazt, egy CSV fájlt vár el, majd a végrehajtás végeztével egy output.txt nevű állományt készít, melyben a kialakult klaszterek centroidjait, valamint az adott csoportba tartozó sorok azonosítóit lehet végig olvasni. Az adatok Map-ek és List-ek segítségével kerültek eltárolásra a memóriában. A megoldás tartalmaz egy külön Centroid osztályt, mely a középpontokat reprezentálja, valamint a továbbfejlesztést elősegítendő a távolság számítás egy külön kiemelt interfészt kapott, így az ezt megvalósító osztályok cserélhetők a távolság legjobb kiszámítása végett.

Jelen implementáció esetén az üres klaszterek nem kerülnek bele az output.txt fájlba, így az tapasztalható, hogy egyes esetekben a kívánt „k” darab csoport helyett „k”-nál kevesebb halmazt ad eredményéül a program végrehajtása. Ez akkor fordulhat elő, ha a felhasznált adatvektorok nagy része azonos mintát, esetleg mintázatokat mutat. Erre nyújthat megoldást, ha iteratívan növelt „k” konstans kerül alkalmazásra és a legpontosabb eredményt adó szám lesz a későbbi futtatások alapja.

A mérések során magának az algoritmusnak a futási idejét kell figyelembe venni, ehhez nem külső eszköz, hanem a kódban a beolvasást követően beillesztett plusz sorok lettek felhasználva, így pontos mérést sikerült kialakítani.



	1000	2000	4000	8000	16000	32000	64000
2 cluster	0,082	0,105	0,172	0,323	0,478	0,510	0,791
3 cluster	0,084	0,114	0,181	0,315	0,487	0,764	1,105
4 cluster	0,092	0,142	0,210	0,318	0,451	0,735	1,365
5 cluster	0,096	0,123	0,219	0,325	0,477	0,631	1,293
6 cluster	0,104	0,127	0,222	0,327	0,479	0,638	1,255
7 cluster	0,111	0,131	0,215	0,330	0,442	0,863	1,307
8 cluster	0,115	0,148	0,251	0,335	0,453	0,970	1,537
9 cluster	0,113	0,157	0,265	0,337	0,477	0,974	1,576
10 cluster	0,112	0,189	0,231	0,345	0,551	1,056	1,689

1. ábra - K-Means - Elapsed Time (max iterations = 1000)

A mérések átlagolással történtek, vagyis, adott „k”, valamint meghatározott sorméret mellett 5 darab mérés átlagideje adja a táblázat celláinak értékét. A diagram azt prezentálja, hogy adott körülmények között milyen teljesítményre, gyorsaságra képes a K-Means algoritmusra épülő program. Kiemelt tényező, hogy a maximális iteráció számnak ezres tényező lett beállítva, ugyanis számottevő változást nem lehetett tapasztalni az ismétlési érték növelése során. Az elkülönítve szereplő oszlop sokaságok jelölik, hogy hogyan viselkedett az algoritmus meghatározott bemeneti sor darabszám mellett. Az adott oszlopok pedig szín szerint elkülönülve mutatják, hogy mennyi időbe telt „k” darab klaszter elkészítése, melyek pontos értékét a táblázat cellái reprezentálják – szekundumban meghatározva.

Következtetésként levonható, hogy még 64000 soros bemenet esetén is meglehetősen gyorsan mindösszesen egy, másfél másodperc alatt eredményre jut a program. Az is megfigyelhető, hogy 2-3 csoportra bontás viszonylag egyszerűbb, hiszen ezek jellemzően kevesebb ideig tartottak. Végül, de nem utolsó sorban látható, hogy a végrehajtás az 5, illetve 6-os „k” esetén lokális minimumokat produkált a számítási időben, ami arra utal, hogy a bemeneti adathalmaz optimális csoportosítási száma az egyik e kettő közül. Az is látszik, hogy az elvárt klaszterek számának növekedése jelentősen növelheti a szükséges végrehajtási időt.

2.1.2 Mean-Shift Clustering

A Mean-Shift Clustering nevezetű csoportosító algoritmus egy csúszóablak (sliding-window) alapú megoldás arra, hogy megtalálásra kerüljenek az adatok sűrűbb területei. A K-Means-hez hasonlóan ez is egy centroidok segítségével működő metodika, aminek a célja az, hogy azonosításra kerüljenek a halmazok középpontjai. Működésének alapja a középpont jelöltek frissítése, amit a csúszóablakon belüli pontok átlagával hajt végre. Ezeket az úgynevezett jelölt ablakokat később szűrni kell egy utó folyamatban, hogy megszüntetésre kerüljenek a közeli elemek ismétlődései. Így alakulnak ki a végső centroidok és a hozzájuk tartozó klaszterek.

A K-Means-szel ellentétben nincs szükség arra, hogy a felhasználó megadja a csoportok számát („k” konstanst). Ezen kívül szintén előnyös az a hozzáállás, hogy a lokális optimumok felé halad az algoritmus a csúszóablakok közepével, hiszen ez jól illeszkedik az adatvezérelt gondolkodásmódhoz. Ellenben komoly nehézséget jelenthet a csúszóablak sugarának („r”) meghatározása, hiszen az nem triviális.

Nagy dimenzió számok esetén az algoritmus komplexitása O($T \times n^2$), ahol „T” az adatvektorok, míg „n” a dimenziók (attribútumok) száma – egyszerűbb esetekre O($T \times n \times log(n)$) komplexitás érhető el.

A következő szakasz az algoritmus fő elemeit ismerteti.

Bemenet:

- egy adatvektor halmaz (pl. CSV fájl), címkék, attribútum elnevezések nélkül

Lépések:

1. Csúszóablak készítése, melynek sugara és középpontja véletlenszerűen választott.
2. Minden iterációban a csúszóablak a nagyobb sűrűségű régiók felé tolódik el úgy, hogy a középpont az ablakon belüli pontok középértékére kerüljön. Az ablak sűrűsége a benne lévő pontok számával egyenesen arányos.
3. A csúszóablak megáll, ha a sűrűség szempontjából lokális maximumra jutott.
4. Amíg nincs lefedve minden pont, addig 1-3. lépés ismétlése.
5. Ha minden adatvektor fedése megtörtént, akkor a duplikáltan fedett elemek hovatartozásának eldöntése távolság alapján

Kilépési feltétel:

- Minden adatvektor klaszterhez van rendelve.

Kimenet:

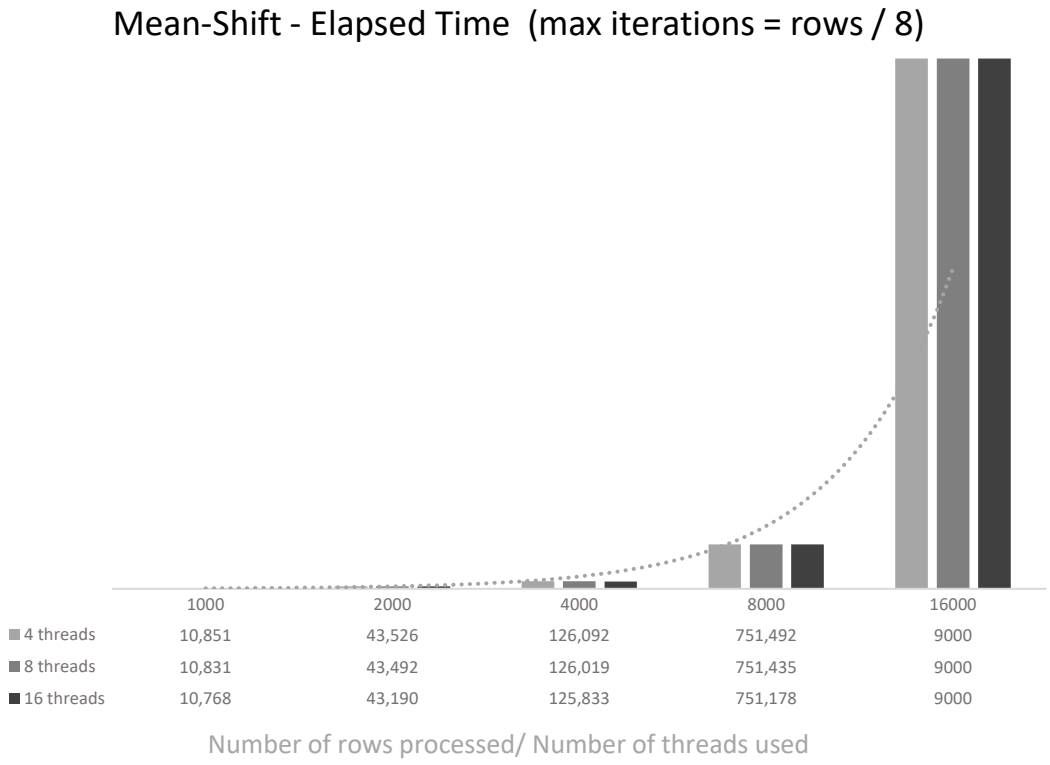
- Egy csoportosítása a bemenetként kapott vektoroknak.

Az alábbiakban a mérési eredmények kerülnek bemutatásra.

A készített prototípus megvalósítása során az alap Mean-Shift algoritmust egy külső könyvtár biztosítja. A könyvtár (library) neve Java Statistical Analysis Tool, röviden JSAT, ami egy tisztán Java programozási nyelven írt. Ez az egyik algoritmusok szempontjából leggazdagabb keretrendszer ezen a nyelven. Más hasonló gépi tanulással foglalkozó könyvtárakhoz képest a mérések alapján gyakran gyorsabb. Rendelkezik K-Means, AdaBoost és RandomForest implementációkkal is.

A könyvtár használata során egyszerűen lehet fájlból adatokat beolvasni, viszont a sima CSV fájlokat át kell alakítani ARFF (Attribute-Relation File Format) formátumú állományokká az előfeldolgozás során a hatékony működés végett. Ez egy olyan formátum, amit adatbányászathoz szoktak használni. Két jól elkülönülő szekcióra lehet bontani. Az egyik rész az úgynevezett „Header”, vagyis, hogy az adott tábla milyen

attribútumokat fog tartalmazni, és ezek milyen típusúak. A második szakasz az úgynevezett „Data” szekció, ahol maguk az adatok kerülnek tárolásra a CSV-khez hasonló módon. Az eredeti fájl beolvasás után az előfeldolgozás eredményeként csupán azon attribútumok kerülnek bele az ARFF állományba, amelyek megfelelő differenciáltságot adnak, így a havi összfogyasztási adatok alkotják az új bemenetet. Szintén a preprocesszálás feladata, hogy a szám típusú mezőkben tényleg számok szerepeljenek, illetve, hogy a „null” értékek 0-ként szerepeljenek. A mérés során a korábbiakhoz hasonló módon lett végrehajtási idő meghatározva.



2. ábra - Mean-Shift - Elapsed Time (max iterations = rows / 8)

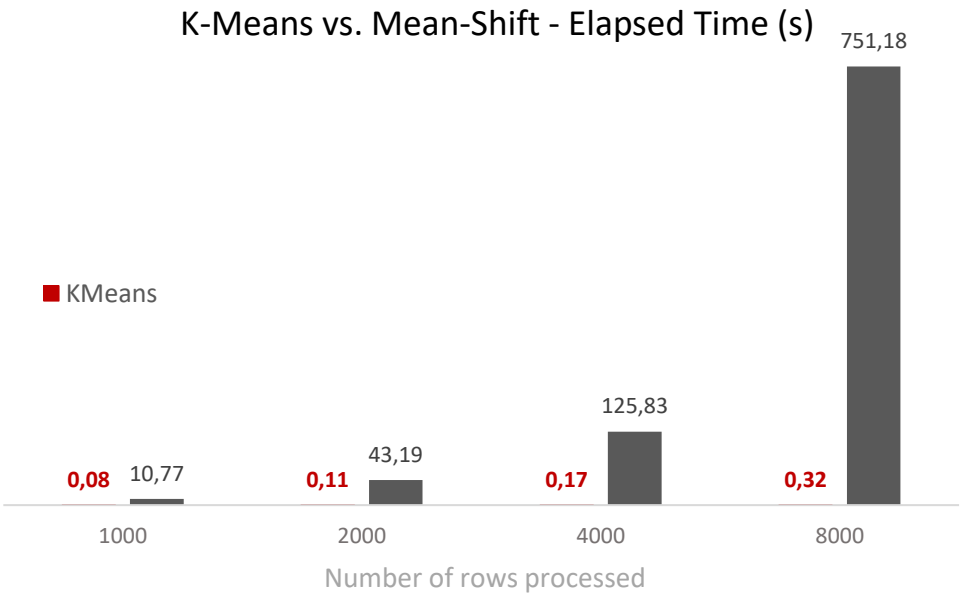
A teszteléshez használt számítógép négy maggal és ezzel megegyező számú logikai processzorral rendelkezett. Ez magyarázza azt, hogy a 4, 8, 16 szállal történő futtatási eredmények nem mutatnak nagy eltéréseket. Azt fontos megjegyezni, hogy 2 szál esetén a végrehajtás túl sokáig tartott, így azzal a tesztelés végül nem lett végrehajtva. Szintén érdekesség, hogy az algoritmus komplexitásából adódóan 8000 sor felett a végrehajtási idő már több, mint 2 és fél órát vett volna igénybe, így ezen kísérlet félbeszakadt. Ezt jelöli a 9000-es szám $(60\,min \times 60\,sec \times 2.5 = 9000)$, de a végeredményből így is le lehet vonni a következtetéseket. Továbbá az is jól látható, hogy míg a K-Means algoritmus exponenciálisan növekvő sorokra exponenciálisan nő, addig a Mean-Shift ugyanezen sorra láthatóan bőven az exponenciális görbe fölé esik, ami igazolja a komplexitásából adódó elvárásokat. Az elkészült „mean-shift_proto” program

a „kmeans-proto” alkalmazáshoz hasonlóan ugyanazzal a CSV fájllal dolgozott, illetve a kimenet is szinte azonosan néz ki, ami jelen esetben is egy output.txt állományba kerül.

Összehasonlítható a K-Means algoritmussal az a kialakított klaszterek száma. Míg az imént említett módszer 2-10 darab csoporttal lett tesztelve és gyakran 5-nél több nem üres klaszter nem is keletkezett, addig a Mean-Shift esetében – ahol az algoritmus maga dönti el, hogy hány csoport jöjjön létre – kisebb bemeneti sorszámok esetén – pl. 1000 sorra körülbelül 40 halmaz lett előállítva, viszont 4000 sor esetén a megoldás már egyre inkább közelített a korábban tesztelt metodikához, 6-7 klaszter került csak létrehozásra.

2.1.3 Klaszterező algoritmusok összehasonlítása

A Mean-Shift komplexitásból adódó problémái és a nehezebb bővíthetőség / javíthatóság végett, a K-Means használata a javasolt az energetikai szektorhoz kapcsolódó hatalmas méretű, úgy nevezett „BigData” rendszerekhez. Ugyanakkor megfontolandó egyes, kevésbé összetett felhasználási esetekben a Mean-Shift használata, hiszen nagy előny, hogy nem kell megadni számára a kívánt klaszterek számát, valamint a kiinduló centroidokat sem és így is egy általánosságban megfelelő csoportosításra jut. A csoportszámok megadásának hiánya hátrány is lehet, mert megfelelő választás esetén a jó kezdeti középpontokkal jobb csoportosítást lehet elérni, illetve hierarchikus algoritmus futtatással a K-Means végrehajtása során sem szükséges a csoportok számát előre definiálni. Szintén hasznos lehet időközönként a K-Means mellett futtatni, mert ezzel ellenőrizhetjük, hogy mennyiben térnek el a két algoritmus által kialakított halmazok, és ez visszacsatolást jelenthet, hogy minden rendben működik-e.



3. ábra - K-Means vs. Mean-Shift - Elapsed Time (s)

2.2 Predikciós algoritmusok

A deep learning algoritmusok egy nagy csoportját jelentik az úgynevezett az osztályozó vagy regressziós módszerek. Az adatok megfelelő csoportosítása elégséges alapot nyújthat a predikció végrehajtásához, ugyanakkor az is előfordulhat, hogy olyan megközelítés kerül alkalmazásra, mint amilyen a korábbi időjárási adatok és az ahhoz kapcsolódó napenergia termelési értékek. A kiindulási metaadat azért szükséges, mert a predikciós metodikák felügyelt környezetben működnek. Ez azt jelenti, hogy szükséges egy betanításnak hívott fázis, ezt követően lehetséges – a már kialakult felépítéssel – a jövőre/ismeretlenre vonatkozó becslések kialakítása.

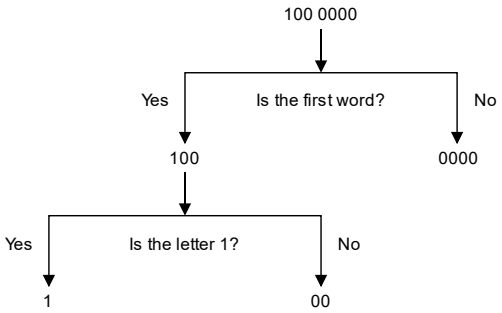
2.2.1 Random Forest

A Random Forest algoritmus képes osztályozó, valamint regressziós működésre is. Az osztályozás kifejezést eldöntendő, „igen-nem” kérdések esetén szokás használni, míg a regresszió elnevezés a több számot, jósolni képes, például a napelem hatásfokát előrejelző megoldásra használandó.

A Random Forest az úgynevezett Bagging (Bootstrap Aggregating) egyik tovább fejlesztett változata, így fontos előbb ennek a működését megérteni. A Bagging algoritmus az angol „bag” szóra is apellál, hiszen kezdetben „m” darab zsákot, halmazt kell kialakítani. Ezt követően az „n” darab tanító adatot fel kell osztani a csoportokba, véletlenszerű módon, úgy, hogy lehetőség legyen felülírásra, ha például egy elem már korábban bekerült az adott halmazba. A csoportokból a tanulás során így úgynevezett modellek jönnek létre. Az éles (nem tanuló) működés úgy zajlik, hogy az adott adatvektor bekerül mind az „m” darab modellbe, és az adott modellek jóslatait figyelembevéve születik meg a végeredmény. Ez a metodika leginkább akkor hatékony, ha magas az adathalmaz varianciája („variance error” – szórás hiba, az adathalmaz ingadozásával szembeni érzékenység, a túl magas variancia akár a zaj modellezését eredményezheti), és alacsony a torzítási hibája („bias error” – a tanulási algoritmus téves feltételezései, amik elrontják a bemenetek és a hozzájuk tartozó célkimenetek összekapcsolását). A Bagging esetén tehát egy modell helyett „m” darab kisebb, különböző modell adja a kimenetet.

A Decision Tree (döntési fa) ismerete szintén szükséges a Random Forest megértéséhez, hiszen az „erdő” ilyen fákból fog felépülni. Egy egyszerű példa látható a következő ábrán. Az adathalmazt szét lehet vágni aszerint, hogy a benne lévő számok az első szóba tartoznak-e vagy sem. Az első szóbeli betűkről el lehet dönteni, hogy azok 1-

es számok-e vagy sem. Ezek alapján elmondható, hogy a fa csomópontjai az úgynevezett vágási pontok lesznek.



4. ábra - Döntési fa példa

A Random Forest több döntési fából építkezik, innen ered az erdő elnevezés. A bemenő adatokra minden fa elő fog állítani egy jóslatot az adott adatvektor hovatartozását illetően, majd a végső predikció az lesz, amit a legtöbb fa (Bagging-re asszociálva: modell) javasol. Az alapgondolata az algoritmusnak a „tömeg bölcsessége" – ez azért működik, mert nagyszámú relatíve különböző, nem korreláló modell, mint egy tanács képes döntést hozni, ami szinte mindig nagyobb hatékonyságot mutat, mint egyetlen önálló, komplex modell. Érdemes megfontolni azt, hogy így általánosságban egy-egy fa véletlen tévedését a többi modell képes kiküszöbölni. A Random Forest algoritmus tehát két fontos előkövetelménnyel rendelkezik. Az egyik, hogy lehessen választani az attribútumok mentén, olyan vágásokat, amelyek eltérnek a véletlenszerű találgatásoktól. A másik, hogy a kialakult fák egymáshoz viszonyított korrelációja legyen alacsony.

A következő szakasz a Decision Tree algoritmus fő elemeit ismerteti.

Bemenet:

- Adathalmaz, amiben több attribútumból álló sorok vannak (tanítók, és tesztelők).

Lépések:

1. Ki kell választani a legjobb attribútumot, ez lesz a fa gyökér csomópontja.
2. Szét kell vágni az adatokat két alcsoportra az adott attribútum mentén. Ezt úgy kell előállítani, hogy a kialakított egységek az adott attribútumot tekintve a kritérium szerint egységesek legyenek.
3. Az 1-2. lépéseket meg kell ismételni egészen addig, amíg a fa minden ága levélcsomóponttal végződik (olyan csomópont, ami egy besorolási osztályra utal).

Kimenet:

- Egy döntési fa, a levelek egy-egy kimeneti osztályt adnak.

A következő szakasz a Random Forest algoritmus fő elemeit ismerteti.

Bemenet:

- Adathalmaz, amiben több attribútumból álló sorok vannak (tanítók és tesztelők).

Lépések:

Tanítás:

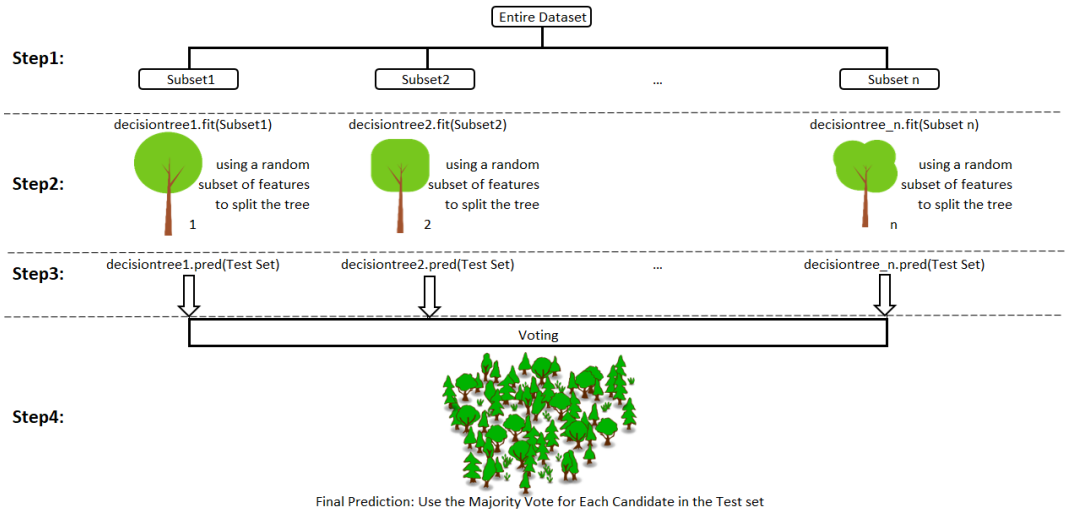
1. Ki kell választani véletlenszerűen „k” elemet az összes „m” elemből, ahol „k” nagyságrendekkel kisebb kell legyen, mint „m” (k << m).
2. A legjobb vágás alapján ketté kell osztani az adatvektorokat. Maga a vágás egy csomópontot alkotni.
3. A létrejött adathalmazok lesznek a gyermek/leszármazott csomópontok.
4. Az 2-3. lépéseket ismételni kell az új csomópontokra, amíg a döntési fa elkészül.
5. Az 1-4. pontok ismétlésével erdőt, azaz „n” darab döntési fát kell készíteni.

Predikció:

1. Az aktuális teszt adatvektort kell venni, majd az összes véletlenszerűen létrehozott döntési fával meg kell határozni a hozzátartozó kimeneti osztályt.
2. Ki kell számolni javasolt osztályokra az előfordulási számot (osztályok maguk a jóslatok, feltehetően több fa is fogja ugyanazon osztályt javasolni).
3. A legnagyobb számú javaslat lesz az adott sor végső predikciója.
4. Az adathalmazon ismételni kell az 1-3. lépéseket a fennmaradó sorokra.

Kimenet:

- Tanítás esetén egy döntési fákból álló erdő.
- Predikció esetén az eredeti teszt adathalmaz kiegészítve a javaslattal.



5. ábra - Random Forest algoritmus szemléltetése [5] (nagyobb ábra mellékletben található)

Az alábbiakban a mérési eredmények kerülnek bemutatásra.

A prototípus elkészítése során használt keretrendszer azonos a Mean-Shift algoritmus során felhasználttal. A könyvtár (*library*[1]) neve Java Statistical Analysis Tool, röviden JSAT, ami egy tisztán Java programozási nyelven írt könyvtár. Működése során legkönnyebben – speciális –, úgynevezett ARFF formátumú állományokat képes kezelni, melynek érdekességei a Mean-Shift prototípusához tartozó részénél lettek ismertetve. Az információhalmaz egy olyan 32688 soros CSV fájl, ahol időjárási adatok, így a légnyomás, páratartalom, hőmérséklet, napfelkelte ideje, napnyugta ideje, és az aktuális idő került eltárolásra, mindamellett, hogy ismert volt, hogy adott körülmények között a kiválasztott időpontban mekkora a napsugárzás ereje. Ez az attribútum lett cél tulajdonságként kiválasztva, mert ebből számolható a napelemek hatékonysága, s így, hogy mennyi energiát kell a szolgáltatónak az adott területre eljuttatnia.

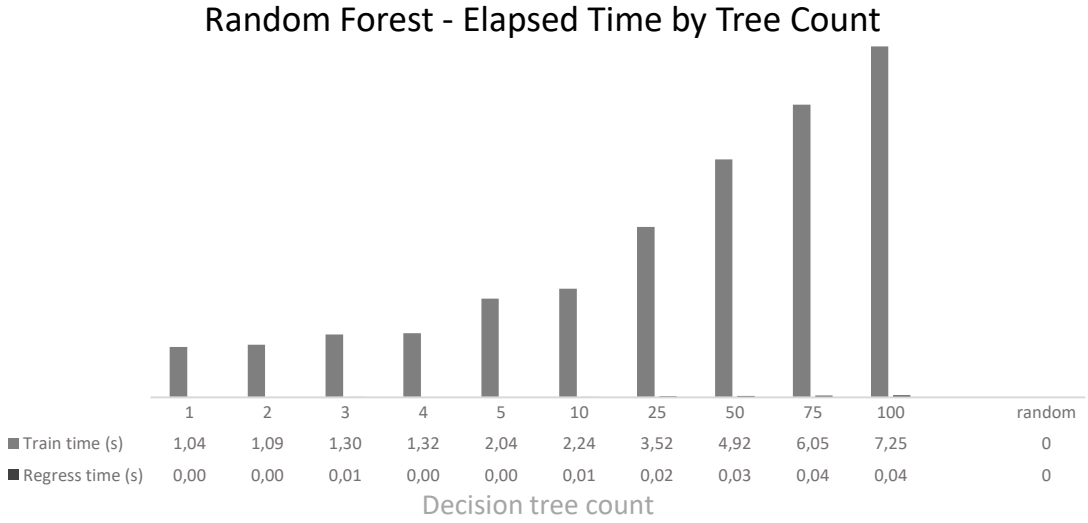
A bementi CSV formátumú adathalmaz, míg a kimenet egy „output.txt” elnevezésű állományba kerül, a korábbi prototípusokhoz hasonló módon. A tesztelés során kiderült, hogy – az elméleti anyagoknak megfelelő módon – a kis varianciával rendelkező attribútumok ronthatnak a módszer hatékonyságán, így a napkelte, napnyugta ideje, továbbá a pontos UNIX idő és a dátum az ARFF formátummá konvertálás során kikerültek figyelembe vett tulajdonságok közül. Szintén adatkonvertálás témakörébe tartozik, az, hogy szükséges az adatokat a Random Forest számára „fogyaszthatóan tálalni”. Ez azt jelenti, hogy a nem számformátumú mezőket numerikussá kell alakítani. Ennek egyik módja a jelen esetben is alkalmazott megoldás, miszerint az összes nem szám formátumú karaktert, kivétel a tizedesvesszőt/-pontot el kell távolítani.

A működés során megadható az erdő maximális nagysága, azaz, hogy hány darab modell jöjjön létre legfeljebb a tanítás során. A regressziós végrehajtás a teszt vektorokon egyesével hívható, így a predikció akkor kapható meg, ha egy lista kerül végig iterálásra. A mérések a korábbiakhoz hasonlóan – külső hívás nélkül – néhány plusz kódsor beszúrásával, pusztán az algoritmus futását – a fájl műveleteket nem – mérve, történtek. A prototípus kimenete két féle mérőszámot tartalmaz az algoritmusra vonatkozóan. Az átlagos eltérés, vagy másnéven átlagos hiba kiszámítása úgy történik, hogy a teszt adatsorok esetén adott volt az elvárt érték, és a kapott jóslat. A két érték eltérése kerül összeadásra minden egyes rekord esetén, majd ezeket az összes elem számával elosztva kapható az eredmény az aktuális végrehajtásra. Az MSE („Mean Squared Error”)

[1] https://github.com/EdwardRaff/JSAT/wiki

jelentése átlagos négyzetes eltérés. Kiszámítása a következő módon történik: $MSE = \frac{1}{n} \times \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, ahol „$y_i$" az elvárt, a „$\hat{y}_i$" pedig a predikció értékét jelöli. Ezen metrikák a további diagramok esetén kerülnek részletezésre.

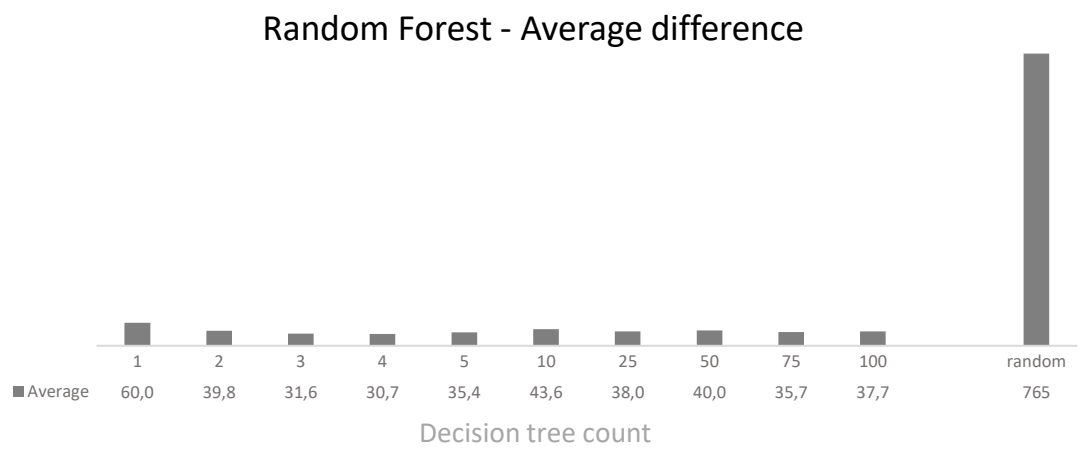
A diagramok rendre tartalmaznak egy-egy oszlopot „random", vagyis véletlen felirattal jelölve. Ez azért van, mert így jól szemléltethető, hogy az algoritmus mennyivel tud hatékonyabban működni, mint a teljesen véletlenszerűen az adott cél attribútum értékkészletéből számokat jósló megoldás. Az idők meghatározása során nem releváns a véletlenszámok generálásának ideje, így ez nullának tekinthető, a mérésre nincs hatással.



6. ábra - Random Forest - Elapsed Time by Tree Count

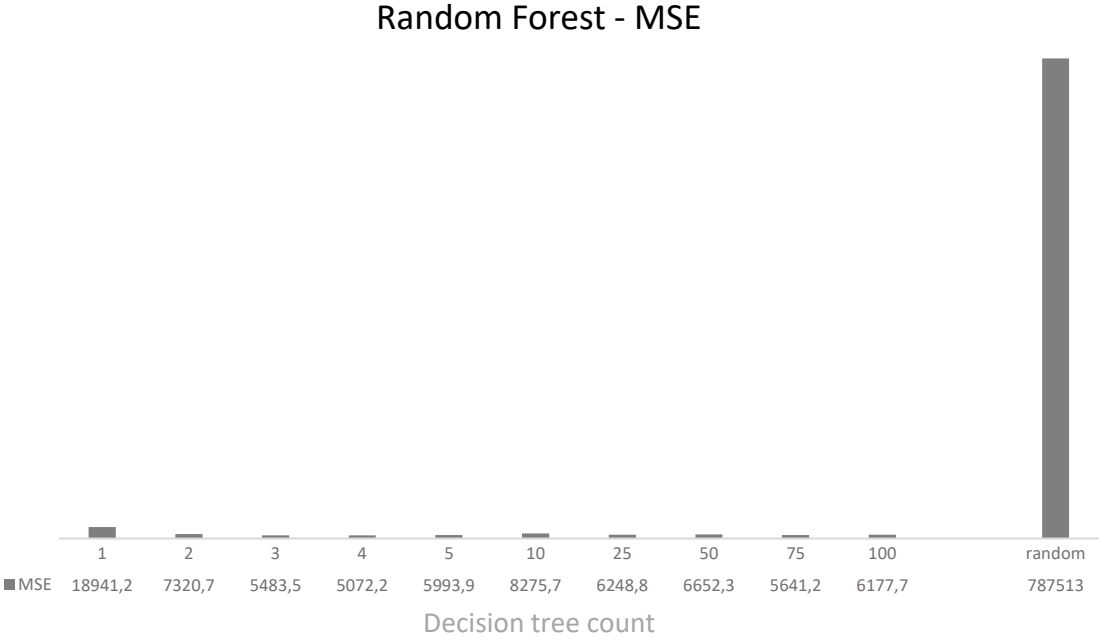
Az első diagram a RandomForest algoritmus tanítási és regressziós idejét szemlélteti, akár 150 döntési fát magába foglaló „erdők" esetében is. Megfigyelhető, hogy maga a regresszió nagy méretű „erdő" esetén sem tart túl sokáig, viszont itt fontos figyelembe venni azt is, hogy a tesztelő sorok száma nagyságrendekkel kisebb a tanító rekordok mennyiségénél, végső soron tanítani viszont csak egyszer kell, így ez pozitív.

A tanításra vonatkozó idők esetén fontos észben tartani, hogy a vízszintes tengelyen felsorolt „erdő" méretek nem lineárisan növekednek. Ennek megfelelően a legkisebb, csupán egy döntési fából álló modell rendszer betanítási ideje valamivel több, mint egy másodperc, míg a 150 méretű „erdő" ezen végrehajtási lépése közel 11 szekundumig tart. Ebből kikövetkeztethető, hogy a fák számának növelésénél lassabb mértékben növekszik a végrehajtási idő, hiszen míg a tanítás 11-szeresre, addig a fák száma 150-szeresre változott. Ez azt mutatja, hogy a nagyobb méretű adathalmazok esetén is arra lehet számítani, hogy az eredmény javítása érdekében meg lehet fontolni a nagyobb méretű „erdők" alkalmazását, hiszen az nem rontja túlságosan a hatékonyságot.



7. ábra - Random Forest - Average difference

A következő diagram („7. ábra - Random Forest - Average difference”) az adott végrehajtás során jósolt és elvárt értékek átlagos eltérésének összegét ábrázolja. Az figyelhető meg, hogy 3-4 fa esetén kialakul egy optimum, tehát ez a modell szám az ideális ezen tanító adathalmaz esetén. Ennél kevesebb döntési fa még nem tudja kiküszöbölni az egy-egy modellből fakadó hibákat, viszont ennél több már veszélyeztetett a túltanulás problémájával, vagy éppen nem lesz eléggé diszjunkt az újonnan hozzávett fa a korábban meghatározottaktól. A Random Forest jól teljesít átlagos eltérés szempontjából. A teljesen véletlen eredményekhez képest 95%-os az előrelépés, míg a leggyakoribb elemre történő becsléshez képest is 28%-os az előrelépés.



8. ábra - Random Forest - MSE

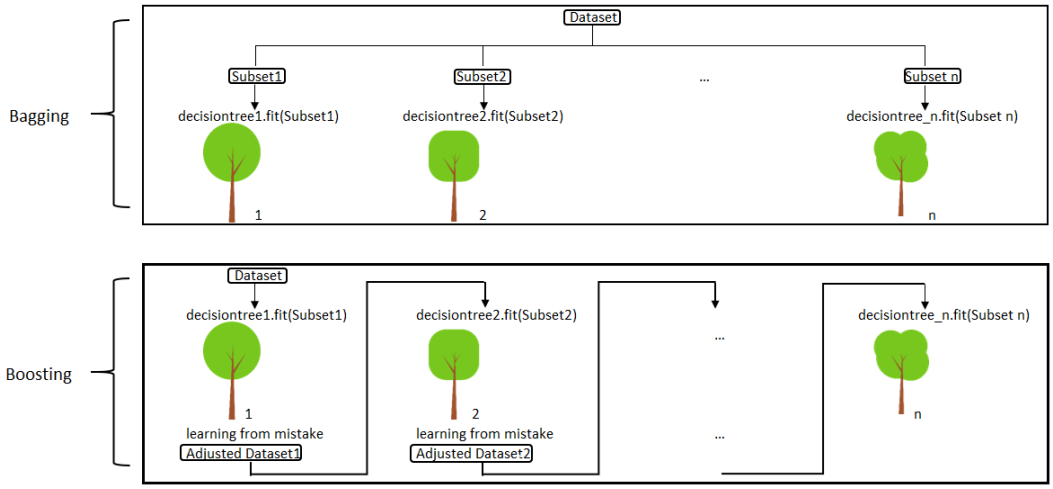
Az utolsó diagram („8. ábra - Random Forest - MSE”) a korábban ismertetett MSE mérésekre vonatkozó adatokat szemlélteti. Ebben az esetben is megfigyelhető az

optimum hely kialakulása a 4 körül, a fák darabszámára vonatkozólag. A RandomForest 99%-os javulás okoz a teljesen random adatokhoz képest, míg a legtöbbször előforduló elemre történő becsléshez képest a javulás 52%-ot jelent, ami szignifikáns.

Összegezve elmondható, hogy a RandomForest egy érthető, jól használható és implementálható regressziós algoritmus, ami hatékonynak és gyorsnak tekinthető, továbbá már viszonylag kisebb modell szám esetén jó értékeket tud produkálni.

2.2.2 AdaBoost (Adaptive Boosting)

A Boosting egy olyan gépi tanulási technika, ami lényegében olyan algoritmusokat foglal magába, amelyek alapvetően gyenge tanulási képeségekkel rendelkező algoritmusok kombinációiból állnak, és ennek eredményeként végül hatékony algoritmusok állnak elő. Míg a korábban megismert Random Forest a már bemutatott Bagging mechanizmusra épített, ami teljesítmény szempontjából azért hatékony, mert a modelleket egyidőben párhuzamosan is ki lehet számítani, addig a Boosting algoritmusok, építkeznek a korábban létrehozott modellekre. Ez azt is jelenti, hogy a Boost metodika egyik fő hátránya a szekvenciális futásból adódó valamelyest alacsonyabb teljesítmény lesz. Ezt szemlélteti a következő ábra.



9. ábra - Bagging és Boosting különbsége [5] (nagyobb ábra mellékletben található)

Az AdaBoost az egyik legelső Boosting metodikára építő algoritmus, továbbá a mai napig is az egyik leghatékonyabb módszer ezek közül. Hasonlóan a Random Forest algoritmushoz szintén arra épít, hogy a sok egyszerűbb modell együttes hatékonysága jelentősen jobb, mint egyetlen önálló komplex modellé. Ez a módszer is a döntési fákat veszi alapul, azzal az egyszerűsítéssel, hogy az adott fa csak egy szintes lehet, ezeket döntési csonkoknak („Decision Stumps”) nevezik – vagyis az első választást követően egyből, azaz osztály adódik, amibe az elemet az adott döntési csonk besorolná. A csonkok

lényegében egy attribútumból és egy küszöbértékből állnak – utóbbi jelenti a vágási feltételt, az értéknél nagyobbak az egyik, a kisebbek a másik osztályba kerülnek, míg a küszöbértékkel megegyezők sorsa implementáció függő. A döntési csonk a lehető legegyszerűbb modell. Azt, hogy egy adott bemeneti adatvektorhoz melyik döntési csonkot célszerű felhasználni, a pontosság alapján lehet meghatározni. Például, ha egy adathalmaz adott attribútuma mindig 1 és az erre az attribútumra épített döntési csonk 60%-os valószínűséggel ad 1-es választ, akkor maga a pontosság is 60% lesz.

A létrehozandó döntési csonkok száma legalább az attribútumok („feature") számával kell megegyezzen. Abban az esetben, ha adott tulajdonsághoz több küszöbérték is meghatározható, akkor ez a szám értelemszerűen többféle is lehet. Azt, hogy ne lehessen végtelen sok küszöbérték, az biztosítja, hogy egy határérték csak akkor alkalmazható, ha az a példákat két jól elkülönülő halmazra bontja. Lényegében a helyes küszöbérték választás a kulcs. A tanító adatokra alapból ismert az elvárt kimenet, így azon elemek, amelyek rossz címkével lettek ellátva nagyobb súlyt (minden tanító rekord súlyozva van, kezdetben a súlyuk 1/n, ahol n a sorok száma) kapnak majd a következő tanulási ciklusban, így a végső küszöbértékek afelé fognak eltolódni, amerre a kimaradt elemek szerepelnek. Ezt bonyolítja, ha több határérték szempontjából is rossz helyre kerül az adott elem, de ekkor a döntési csonkok által osztályozott elemek entrópiája alapján ki lehet választani, hogy melyik legyen az a küszöbérték, amelyet érdemes módosítani.

Az AdaBoost gyakran az itt leírtaknak megfelelően csak két osztályba sorolja a kapott példákat, vagyis ez egy osztályozó algoritmus, ellenben már az eredetileg 2003-ban megjelent leírás szerint is alkalmas volt arra, hogy ne csak bináris problémákra legyen felhasználva – regressziós működés. Ugyanakkor célszerű egy több osztályból álló problémát felbontani kisebb bináris problémákra, és iteráltan alkalmazni a metodikát.

A következő szakasz az algoritmus fő elemeit ismerteti.

Bemenet:

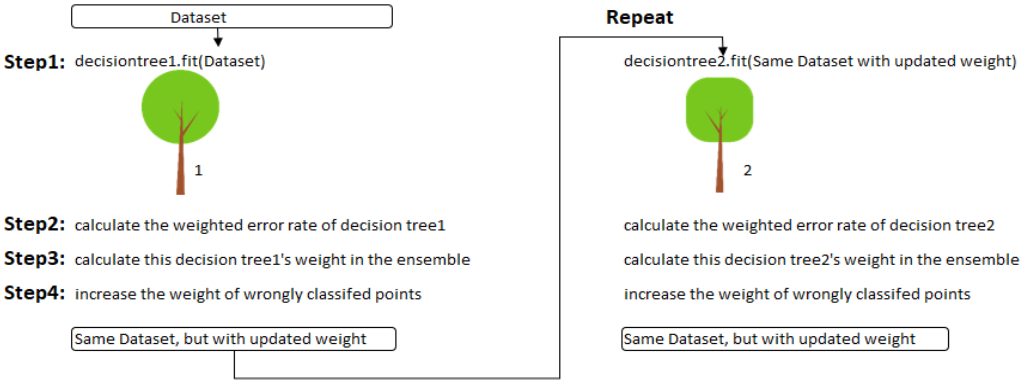
- Egy adathalmaz, amit ketté kell bontani tanító és tesztelő adatvektorokra, illetve mind két esetben ismertnek kell lennie az elvárt kimenetnek.

Lépések:

Tanítás:

1. Létre kell hozni annyi darab döntési csonkot, ahány darab attribútummal rendelkeznek az adatvektorok és ki kell választani egy-egy küszöbértéket.
2. Ki kell osztani a kezdeti súlyokat, minden adatvektor kezdetben 1/n -es súlyt kap.

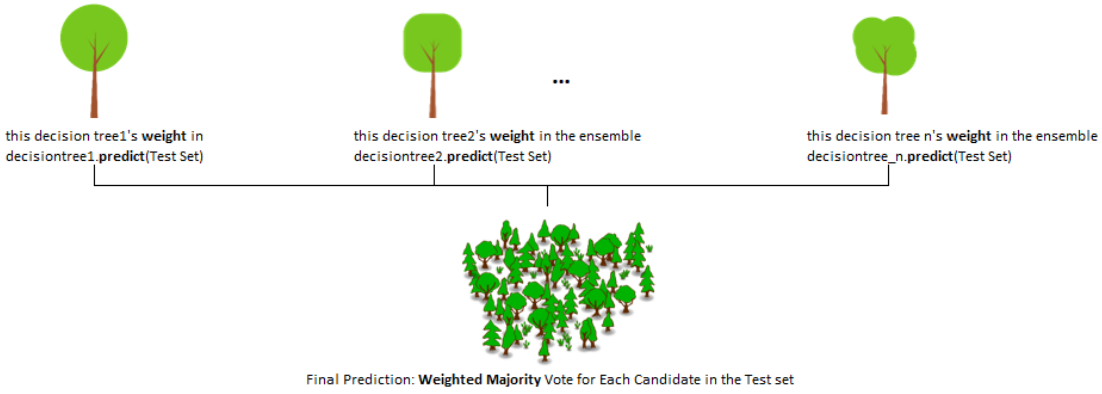
3. Át kell adni a súlyozott adatokat a modelleknek (első iterációban csak az elsőnek).
4. Azon elemeknek, amelyek rossz osztályba kerültek, meg kell növelni a súlyát.
5. Ha van rossz osztályba levő adat, ismételni kell a 3-4. lépést, eggyel több modellre.



10. ábra - AdaBoost tanítás szemléltetése [5] (nagyobb ábra mellékletben található)

Predikció:

1. Venni kell egy teszt adatvektort, majd az összes véletlenszerűen létrehozott döntési csonkkal meg kell határozni a hozzátartozó kimeneti osztályt.
2. Ki kell választani a legpontosabb csonkot.
3. A legpontosabb vagy letöbb csonk által prediktált érték lesz a végső javaslat.
4. Az adathalmazon ismételni kell az 1-3. lépéseket a fennmaradó sorokra.



11. ábra - AdaBoost predikció szemléltetése [5] (nagyobb ábra mellékletben található)

Kimenet:

- Tanítás esetén egy döntési csonkokból álló összetett döntési modell.
- Predikció esetén az eredeti teszt adathalmaz kiegészítve a javaslattal.

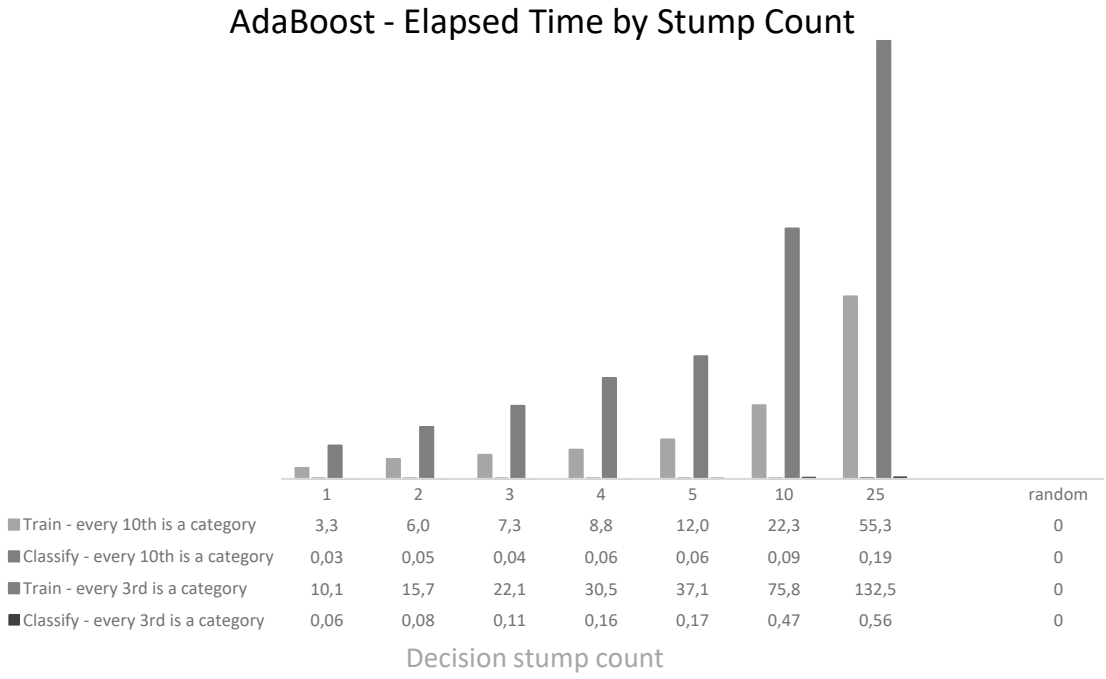
Az alábbiakban a mérési eredmények kerülnek bemutatásra.

Az adathalmaz megegyezik a Random Forest esetén ismertetett forrással, melyből azonos előfeldolgozási okok miatt csak bizonyos attribútumok lettek figyelembe véve – csak ezek diverzifikáltsága megfelelő. A kimeneti „output.txt” állomány és a benne megjelenő átlagos eltérés, illetve MSE kiszámítása a korábban mutatott megoldást követi.

Ezenfelül a felhasznált könyvtár, mely a Java Statistical Analysis Tool, röviden JSAT nevet viseli hatékonyabb működésre képes egy – speciális –, úgynevezett ARFF formátumú segítségével, melynek érdekességei a Mean-Shift prototípusához tartozó részénél lettek ismertetve. Ennek a kielégítése végett az előfeldolgozás során a bemeneti adatokat tároló fájl formátuma is változtatásra kerül.

Az AdaBoost, elsősorban bináris klasszifikálási problémák esetén használatos, ugyanakkor felhasználható multi osztályozási mechanizmusokhoz kapcsolódóan is. Ekkor célszerű lehet annak az opciónak a megfontolása, hogy ne egy összetett predikciós feladat kerüljön véghezvitelre, hanem több binárissá bontott alproblémáé. Ehhez kapcsolódóan a JSAT könyvtár tesz egy ajánlást, hogy az algoritmust együtt lehet használni a „OneVSAll" osztállyal, ahol lehetőség van arra, hogy a probléma automatikusan felbontásra kerüljön a tanítás során megadott predikciós osztályokra épített különböző bináris AdaBoost feladatokra.

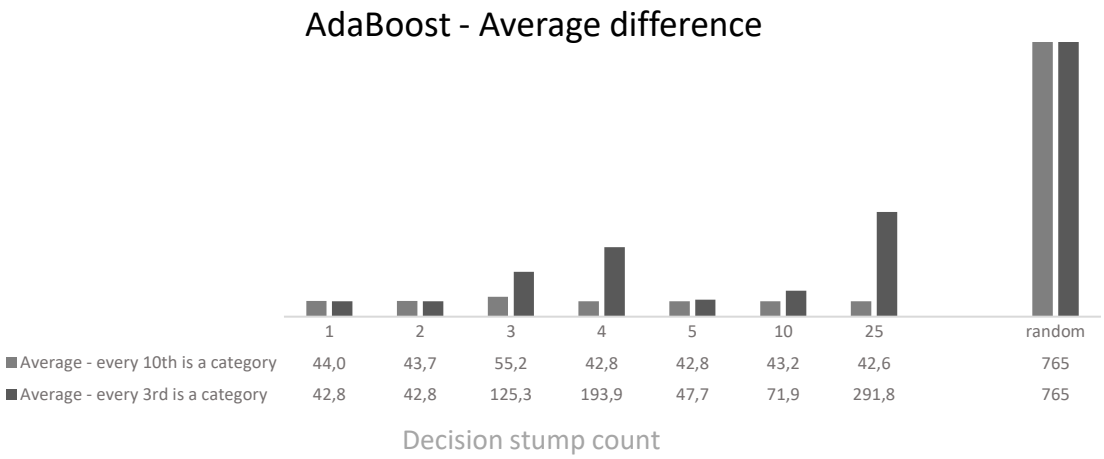
A diagramok az eddigieknek megfelelően rendre tartalmazni fognak egy-egy „random", vagyis véletlen felirattal jelölt oszlopot. Az idők meghatározása során nem releváns a véletlenszámok generálásának ideje, így ez nullának tekinthető.



12. ábra - AdaBoost - Elapsed Time by Stump Count

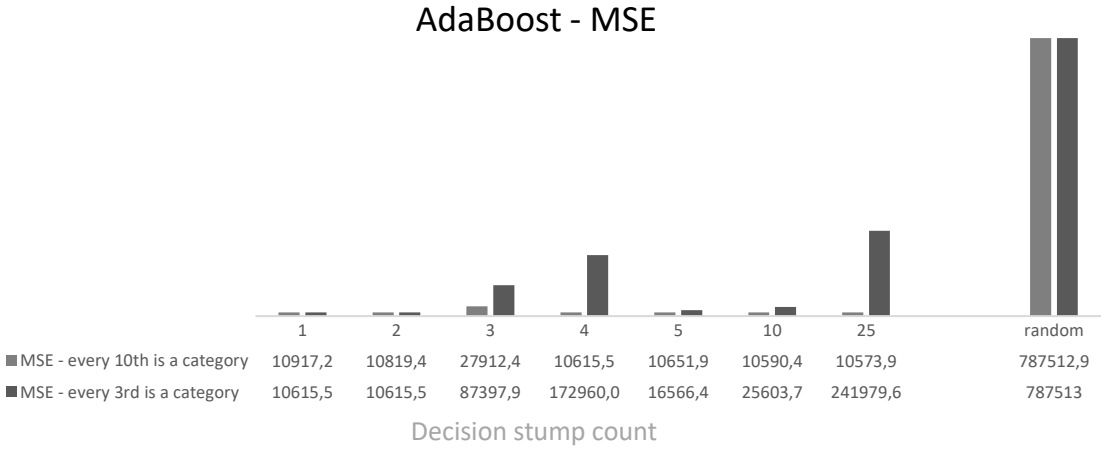
Az AdaBoost prototípushoz kapcsolódó első diagram a tanításának és a predikciós műveletek végrehajtásainak idejét foglalja magába. Az első futtatás során, a döntési csonkok száma alapján kapott eredmények, úgy készültek, hogy a tanító adatok célattribútumának értékkészletének legmagasabb és legalacsonyabb értéke között minden

kerek 10-es érték jelentett egy osztályt, és a tanító számok is 10-esre lettek kerekítve. A második végrehajtási csoport úgy került kialakításra, hogy 10-es helyett minden 3. szám került kiválasztásra, és a tanító értékek is 3-masokara lettek kerekítve. Ez a fajta eljárás bármely adathalmaz esetén alkalmazhatóvá teszi az AdaBoost módszert. Az osztályozási idő nem számottevő, ellenben a tanítás már ekkora adathalmazra is jelentősen megnőtt.



13. ábra - AdaBoost - Average difference

A fenti grafikon („13. ábra - AdaBoost - Average difference") az átlagos eltérését mutatja be. Jól látható, hogy 5-25 darab döntési csonk esetén viszonylag jobban teljesít az algoritmus, viszont – ezt a diagram már nem mutatja be, de – innentől kezdve minél több almodell állt rendelkezésre, annál pontatlanabbá vált a módszer. Legjobb esetben a teljesen véletlen eredményhez képest ugyan jól teljesít a megoldás, de ez az eredmény nagyjából megegyezik azzal, mintha a leggyakoribb elemre történne a becslés.



14. ábra - AdaBoost - MSE

A „14. ábra - AdaBoost - MSE" szemlélteti a négyzetes eltéréseket. Az MSE értékekre szintén igaz a diagram előtt megállapított konklúzió, vagyis, hogy jelen adathalmaz esetén a mindent konstans módon a leggyakoribb elemre becslés azonos eredményt képes elérni az átlagos négyzetes eltéréssel.

Mindkét esetben elmondható, hogy a minden 3 érték után történő osztályozási csoport meghatározás túl sűrűnek bizonyult. Mind teljesítményben, mind hatékonyságban alulmaradt a 10-es kategorizálással szemben. Sajnálatos módon jelen adathalmaz esetén ez a metodika nem tekinthető sikeresnek egyetlen esetben sem. Az AdaBoost algoritmus jelen adathalmaz esetén történő alkalmazása nem javallott, viszont számos olyan aspektus létezik, ahol hatékony működésre képes ez a módszer, így mindig érdemes számításba venni.

2.2.3 XGBoost (Extreme Gradient Boosting)

Az Extreme Gradient Boosting egy olyan algoritmus, ami a Gradient Boosting egy tovább fejlesztett változatának tekinthető. Manapság több az adatbányászattal foglalkozó verseny során került ki győztesen más algoritmusokkal szemben. Érdemes először az utóbbi gondolatmenetét megérteni, és utána áttérni a fejlettebb verzióra.

Boosting algoritmusról lévén szó, a Gradient Boosting (GBoost) is úgy működik, hogy alapvetően gyenge tanuló metodikákból épít erős tanulásra alkalmas módszert. Jelen esetben a megoldandó problémára érdemes egy optimalizációs tevékenységként tekinteni, vagyis alkalmas veszteség függvény végeredményét kell optimalizálni a feladat szempontjából. Ahogy az AdaBoost esetében, úgy itt is az alapkoncepció azaz, hogy úgy javítható a kezdeti egy darab modell hatékonysága, ha kiegészítésre kerül egy újabb modellel, ami csökkenti a veszteséget (a predikció tévedéseit) és ezt iteratívan lehet megtenni. Egy jó kilépési feltétel a tanulási folyamat lezárására, ha közel nullára csökken az elvárt eredménytől a kapott számításnak az eltérése. Magának a hibának az elvárt értéktől való eltérését gradiensnek nevezik (valamilyen irány nélküli mennyiség változásának mértéke).

Az algoritmus működése egy konstans generálással kezdődik. Ez az érték lesz az, ami a legkevesebb hibát adja, a nulladik predikció. Ezután kiszámolható, hogy minden egyes tanító pont milyen eltéréssel, gradienssel rendelkezik az adott pontot megelőző kimenetek alapján. Ez úgy határozható meg, hogy egy tetszőleges módon választott hibafüggvénnyel kiszámításra kerülnek a gradiensek. A hiba számításának menete nem visszacsatolt módon történik, mint a neurális hálók esetén, hiszen míg ott a különböző rétegekben egyszerűen lehet módosítást végrehajtani a hiba ismeretében, addig jelen esetben több ismeretlen tényező okozza ennek a módszernek az alkalmatlanságát. Nem egyértelmű például a levelek száma, a vágási pontok, valamint a fák mélysége sem.

A korábban leírt konstrukció azért jó, mert így minden egyes adatforrásban lévő ponthoz adott, hogy megfelelő optimalizációs lépések után, hogyan javult a GBoost betanítása, továbbá így meghatározható az az „irány”, amerre az optimalizálás során el kell tolódni. Szintén előnyös, hogy a veszteség kiszámítására használt függvény megadható, így például regressziós problémához használható a négyzetes eltérés is. Osztályozási feladat esetén pedig egy ahhoz a konstrukcióhoz illő metódus. Jelen esetben nem a korábbi iterációk során épített modellek segítségével kell megjósolni, hogy milyen legyen a következő modell, hanem az összes adatvektor eltérésének ismeretében kell olyan új modelleket konstruálni, hogy a lehető legjobban a gradiensekhez illeszkedjenek.

Az XGBoost a következő „összetevőkkel” rendelkezik még a fentieken kívül.

- A modelleket okos módon „bünteti” (készteti arra, hogy kisebb hibát adjon).
- Arányosan csökkennek a fák levélszámai.
- A GBoost módszertől eltérő vágási pont meghatározó mechanizmus alkalmazása.
- Sebesség fokozó és hely takarékos megoldások bevezetése.
- További randomizáló paraméter használata (variancia növelése).

Az első pont arról szól, hogy XGBoost esetén nem csak a hiba kerül „büntetésre”, hanem az is, ha egy modellt a túltanulás („overfitting”) problémája veszélyezteti. Az ilyen módon történő komplexitás csökkentést például úgy lehet elérni, hogy figyelemmel követi az adott fa tanulási idejét/mélységét, vagy akár elágazási tényezőjét, esetleg ezek kombinációját, de az is előfordulhat, hogy a veszteség függvény társaként bevezet egy új metódust ennek a komplexitásának figyelésére. Az új metódus így bele fog számítani az adott adatvektor gradiensébe, aminek köszönhetően nem csak a valódi hiba, hanem a komplexitás is negatív hatással lesz az adott sor eltérésre.

A felsorolás második eleme azt jelképezi, hogy a fákat úgy tartja karban az XGBoost, hogy amikor túl nagyra nőne a befolyása egy adott fának, vagy éppen túl magasra emeli a varianciát, akkor csökkenti az adott fa leveleinek számát. Egy komplex mechanizmus határozza meg, hogy pontosan melyik fát és hol kell „visszavágni”.

Az alap Gradient Boosting gyakran sokáig tarthat, hiszen minden egyes lépésben végigmegy az összes lehetséges vágáson, hogy megtalálja a megfelelőt. Ezen ok miatt, az XGBoost nem ezt használja, hanem megvizsgálja az adott attribútum eloszlását, és ez alapján választ ki egy-egy pontot arra, hogy vágási kritériumként tesztelje, és kiválassza a legjobbat. Például ez működhet úgy is, hogy választ egy számot, ez legyen „k”, az adatok becsült hisztogramja (metrikusan skálázott tulajdonságok grafikus ábrázolása)

alapján és az adathalmaz értékkészletéből vesz minden „k”-adik elemet, és ezekre kalkulálja ki a vágás hatékonyságát.

A sebesség és a memóriahasználat nagyon fontos mértékegysége egy algoritmusnak. Az XGBoost hatékonyságának növelése érdekében bevezették, hogy a bemenő adathalmazt blokkokra osztják fel, méghozzá memória blokkokra, és ezeket úgynevezett tömörített oszlop („Compressed Column”) formátumba tárolják el. Ez elősegíti a megfelelő vágási pontok megtalálását, ugyanakkor, ha az adott blokkban lévő adatok gradiens statisztikáit kell lekérni, az cache tévesztési problémákhoz vezethet, hiszen ezek továbbra is az eredeti formában kerülnek tárolásra. A cache-ben lévő értékek felhalmozódását – és így a tévesztési gyakoriság növekedését – elkerülendő bevezették az úgy nevezett „Cache Aware” algoritmust az XGBoost részeként, ami a jól ismert puffer készítésre épül. Annak a meghatározása, hogy mi a helyes blokk méret, egy kompley feladat, hiszen egyeztetni kell a lehető legjobb párhuzamosíthatóság igényét a cache tévesztésekkel. Szintén segít az úgynevezett „Block Sharding” használata, ami azt indukálja, hogy az eredményeket különböző diszkekre írja ki az algoritmus, így például 4 lemez esetén 4 blokkot lehet olvasni egyszerre, ami 4-szeres gyorsulást eredményez.

A további véletlenszerűség bevezetése azt hivatott javítani, hogy a létrehozott modellek a lehető legkisebb mértékben korreláljanak egymással, hiszen az XGBoost algoritmus esetén – éppen úgy, mint az AdaBoost esetén – fontos, hogy a fák a lehető legkevésbé függjenek egymástól. Ez elősegíti, hogy a különböző modellek a lehető legjobb döntésre jussanak közösen.

A következő szakasz az algoritmus fő elemeit ismerteti. Az XGBoost fejlesztései pszeudokód szintjén nem jeleníthetők meg, így az algoritmus az eredeti Gradient Boost lépéseit mutatja be.

Bemenet:

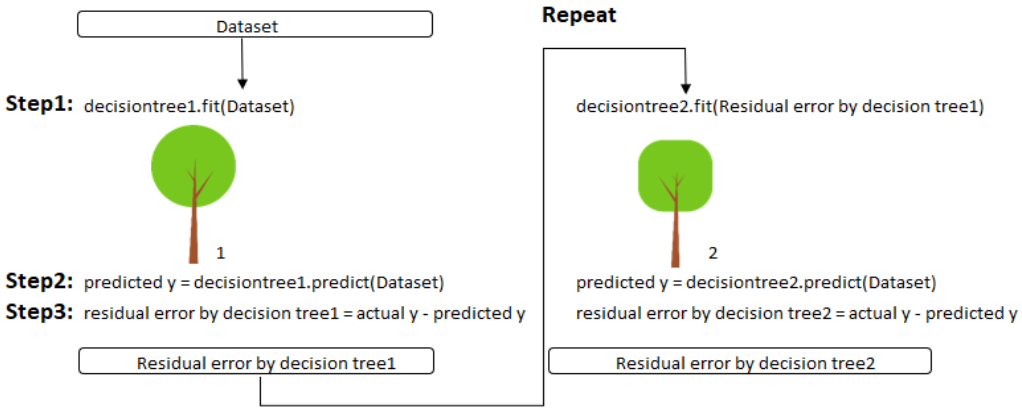
- Egy adathalmaz, amit ketté kell bontani tanító és tesztelő adatvektorokra, illetve mind két esetben ismertnek kell lennie az elvárt kimenetnek.
- Iterációk maximális száma.

Lépések:

Tanítás:

1. Kezdeti konstans kiszámítása.
2. For i = 1-től M-ig (M a modellek maximális száma):

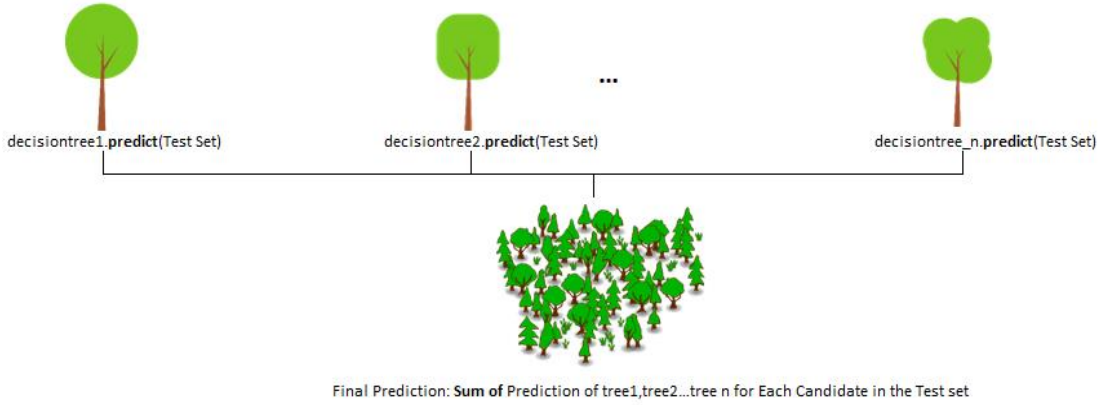
a. Predikciót végre kell hajtani minden adatvektorra.
b. Ki kell számolni az összes adatvektorra a veszteséget.
c. A veszteségből ki kell számolni az összes adatvektorra a gradienst.
d. Gyenge becslő alkalmazása, a modellek frissítésére (gradiensek csökkentése).



15. ábra - Gradient Boost tanítás szemléltetése [5] (nagyobb ábra mellékletben található)

Predikció:

1. Venni kell egy teszt adatvektort, majd az összes modellel meg kell határozni a hozzátartozó kimeneti osztályt, és ezeket el kell tárolni.
2. A végső predikció az egyes modellek által jósolt eredmények összegéből adódik.
3. Ismételni kell az 1-2. lépést, amíg van további adatvektor.



16. ábra - Gradient Boost predikció szemléltetése [5] (nagyobb ábra mellékletben található)

Kimenet:

- Tanítás esetén egy döntési fákból álló összetett döntési modell.
- Predikció esetén az eredeti teszt adathalmaz kiegészítve a javaslattal.

Az alábbiakban a mérési eredmények kerülnek bemutatásra.

Az adathalmaz megegyezik a Random Forest esetén ismertetett forrással, melyből azonos előfeldolgozási okok miatt csak bizonyos attribútumok lettek figyelembe véve – csak ezek diverzifikáltsága megfelelő. A kimeneti „output.txt” állomány és a benne megjelenő átlagos eltérés, illetve MSE kiszámítása a korábban mutatott megoldást követi.

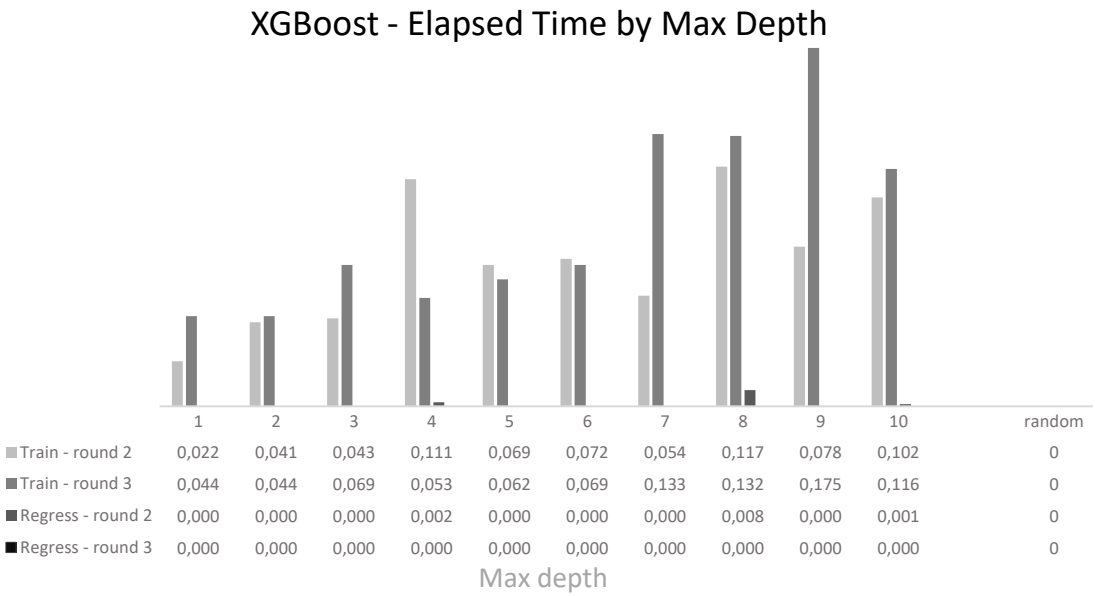
A prototípus elkészítése során az XGBoost4J nevű könyvtár (*library*[1]) került felhasználásra, ahol a „4J” a Java nyelvű implementációra utal. Ebből következik, hogy nem csak ezen a nyelven, hanem más programozási nyelveken is létezik ez a könyvtár, ami alapvetően Python-ban készült, csak előállították más megvalósításait is. A könyvtár kiválasztását olyan tulajdonságok inspirálták, mint, hogy számos nyelven rendelkeznek megvalósítással, továbbá elismert, valamint hatékony és fejlődő könyvtárról van. Kiemelkedő fejlesztéseik közé tartozik a manapság a webes világban egyre inkább megkerülhetetlen, illetve ajánlottan nem megkerülendő klaszterkezelő, ütemező, és még sok más feladatot ellátó Kubernetes elnevezésű eszközzel való együttműködés. [6]

A keretrendszer részletesen paraméterezhető úgy, hogy egy HashMap példányban rögzítésre kerülnek a szükséges tulajdonságok. Jelen esetben az „eta”, vagy másnéven „tanulási ráta” azért felelős, hogy konzervatívabb legyen a tanulás és ne léphessen fel a túltanulás („overfitting”) problémája. Minden Boosting lépés után csökkenti a súlyokat a konzervatívitás elérése érdekében. Alapértéke 0.3, mely jelen esetben 0.465-re lett változtatva a jobb kimeneti eredmények elérése érdekében. A „max_depth”, vagyis „maximális mélység” egy azon adatok közül, mely a mérések során folyamatosan változtatva lett, hogy ellenőrizhető legyen a különböző mélység korlátok esetén a hatékonyság. Elnevezése az algoritmus modelljeiként felhasznált döntési fa mélységére utal. Ezen érték növelése egyre inkább komplexebbé teszi az egyes modelleket. Egy fontos, és rossz tulajdonsága az XGBoost-nak, hogy a mély fák agresszívan fogyasztják a memóriát, így érdemes ezt az értéket körültekintően megválasztani. Alapértéke 6.

A „verbosity”, segítségével az állítható, hogy a naplózás során megszokott szintekhez hasonlóan a tanulás és működés során az algoritmus milyen, valamint mennyi információt írjon ki a konzolra. Érdekes adatokhoz, heurisztikai döntésekhez, lehet így hozzáférni. Az „objective”, azt határozza meg, hogy az XGBoost milyen metodika segítségével működjön. Képes bináris osztályozás és regresszió végrehajtására is, és számos algoritmus elnevezést kaphat ez a paraméter értékül, ami más és más felhasználási eseteket, hatékonyság értékeket ad az algoritmusnak.

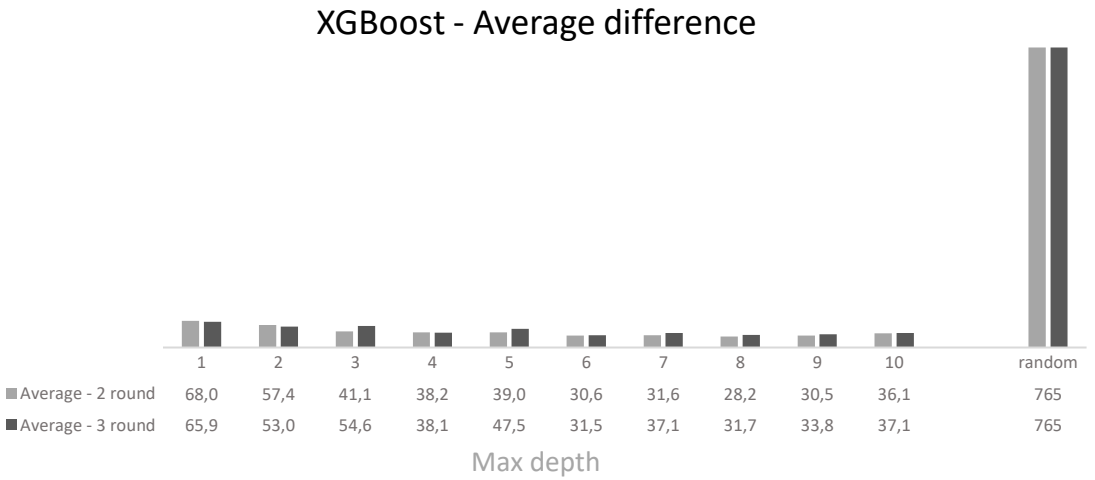
A diagramok az eddigieknek megfelelően rendre tartalmazni fognak egy-egy „random”, vagyis véletlen felirattal jelölt oszlopot. Az idők meghatározása során nem releváns a véletlenszámok generálásának ideje, így ez nullának tekinthető.

[1] https://xgboost.readthedocs.io/en/latest/index.html



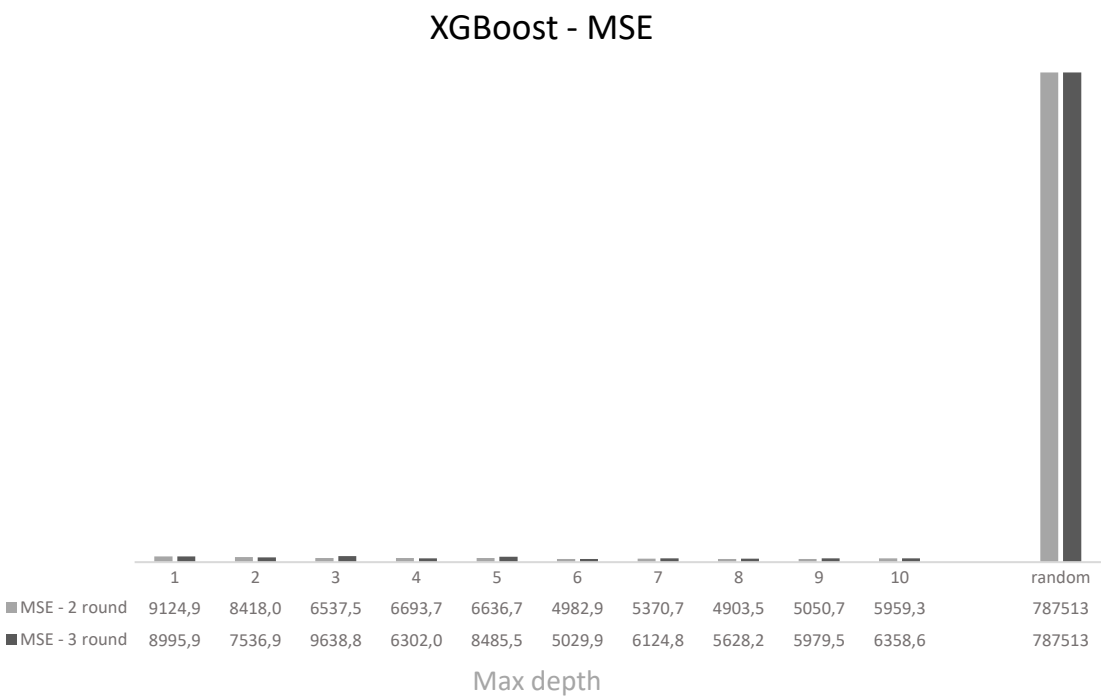
17. ábra - XGBoost - Elapsed Time by Max Depth

Látható, hogy jelen algoritmus esetén a dinamikusan változó paraméter nem csak a döntési fák mélysége, hanem a tanulás során, a tanító pontok lefuttatásának ciklusszáma is volt. A leglassabb tanítási idő sem lépte túl a 0,18 szekundumot, míg a regressziós idők rendre 0,01 másodperc alatt maradtak. A 4-es mélység esetén megfigyelhető kiugró érték arra utal, hogy a maximum 4 mélységű modellek tanítása volt a legrészletesebb.



18. ábra - XGBoost - Average difference

A „18. ábra - XGBoost - Average difference” esetén látható, hogy az optimum a 6-9 intervallumban található, ami a fák maximális mélységét illeti, továbbá a legkisebb eltérés a 8-as esetben figyelhető meg. A teljesen véletlenszerűen jóslatot sorsoló, azonos értékkészlettel (jelen esetben kb. 0-1600) dolgozó véletlenszám generátor eredményeihez képest több, mint 96%-kal teljesít jobban az XGBoost, míg, ha figyelembevételre kerül az az ismeret, hogy mi a leggyakoribb érték, akkor is valamivel több, mint 34%-os a javulás tapasztalható, ami szintén jelentős.



19. ábra - XGBoost - MSE

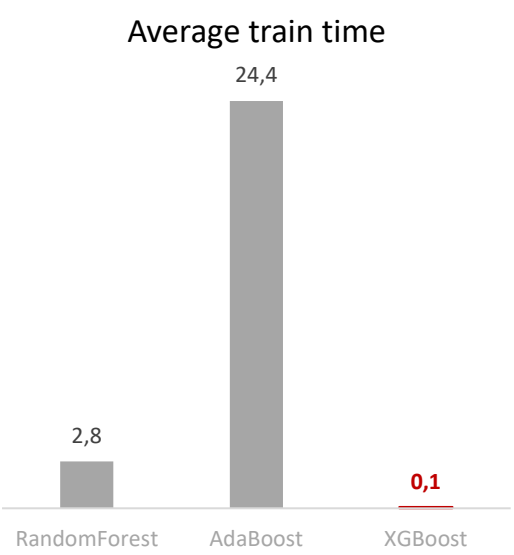
Az XGBoost MSE értékei a fenti ábrán tekinthetők meg („19. ábra - XGBoost - MSE"). Megfigyelhető, hogy az algoritmus ezen értékei is alacsonyak. Ezt a hatékonyságot korábban csupán a RandomForest volt képes megközelíteni, de mint ismert a korábbi felhasznált időkről szóló diagramról, az XGBoost jelentősen gyorsabb, azaz jobb a teljesítménye, valamelyest javított hatásfok mellett.

Az XGBoost a négyzetes eltérés tekintetében több, mint 99%-os javulást hozott. Abban az esetben, ha a leggyakrabban előforduló érték kerül jóslásra az összes teszt sor esetén, akkor 53%-ot meghaladó javulás az eredmény. Ez jól mutatja, hogy az egyszerű heurisztikák és a véletlenszerűen vett értékek mennyivel rosszabbak egy ilyen jellegű gépi tanulási algoritmusnál.

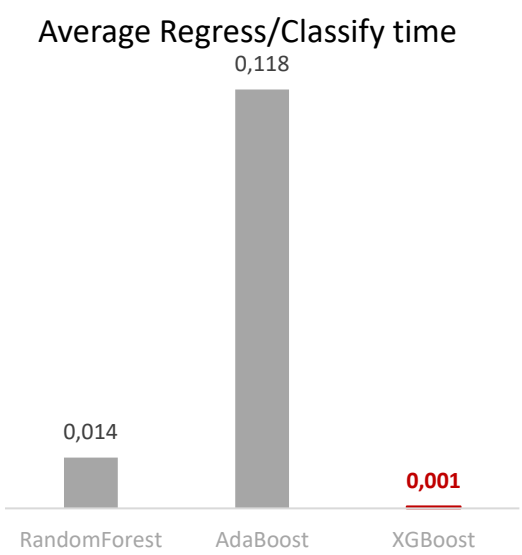
2.2.4 Predikciós algoritmusok összehasonlítása

Az elkészített prototípusok a korábbi fejezetekben bemutatásra kerültek. Mind három kidolgozott predikciós algoritmus ugyanazon adatforrást használta fel a működése során. Az információhalmaz egy olyan 32688 soros CSV fájl, ahol időjárási adatok, így a légnyomás, páratartalom, hőmérséklet, napfelkelte ideje, napnyugta ideje, és az aktuális idő került eltárolásra, mindamellett, hogy ismert volt, hogy adott körülmények között a kiválasztott időpontban mekkora a napsugárzás ereje. Ez az attribútum lett cél tulajdonságként kiválasztva, mert ebből számolható a napelemek hatékonysága, s így, hogy mennyi energiát kell a szolgáltatónak az adott területre eljuttatnia.

A predikciós algoritmusok, amelyek e dokumentumban kidolgozásra kerültek, mind több tanulási módszert építettek egymásra, így „deep learning” metodikáknak tekintendők. Mivel ezek a klaszterező algoritmusokkal ellentétben úgy működnek, hogy a kiértékeléshez használt modell(eke)t először be kell tanítani, így a forrás adatokat két részre kellett bontani.



20. ábra - Predikció: Average train time

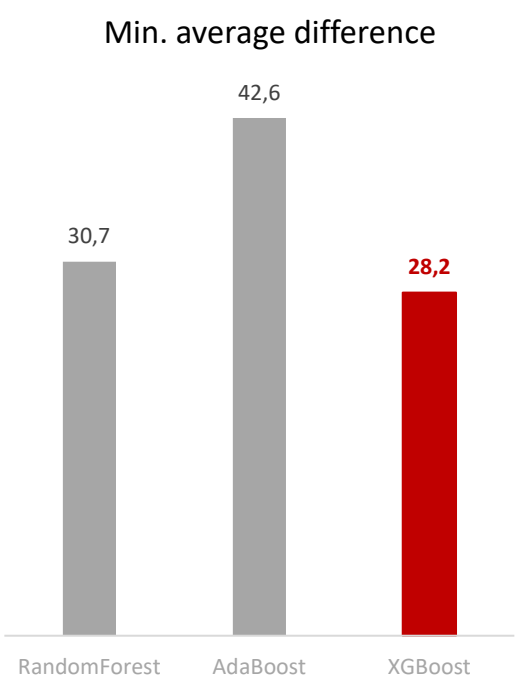


21. ábra - Predikció: Average Reg./Classify time

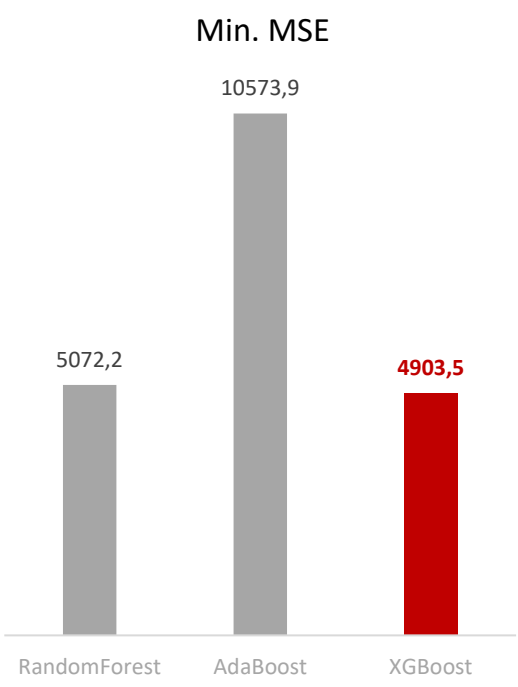
A fenti két diagram a tanulási és regressziós/osztályozási idők szempontjából átlagos adatokat vonultat fel a RandomForest, AdaBoost és az XGBoost esetén. Fontos megjegyezni, hogy időkről (szekundum) lévén szó az az előnyös, ha minél kisebb az eredmény, azaz a megjelenített oszlop. Jól látható, hogy az AdaBoost algoritmust jelen dokumentumban bemutatott felhasználási esetre hatékonyság szempontjából nem volt megfelelő, hiszen nagysárendekkel rosszabb eredményeket adott a másik kettő metodikával szemben. Az AdaBoost esetén érdemes arra is felhívni a figyelmet, hogy ez alapvetően osztályozási („igen/nem döntési”) feladatokra használható leginkább, és bár alkalmas regresszióra is, javasolt helyette „k” fajta eredmény csoport kialakítására, „k” darab AdaBoost rendszerre felbontani a problémát, a már ismertetett módon. Ez egyébként okozhatja a nagyságrendi eltérést, viszont természetesen az Ipar 4.0 alkalmazásai esetén előfordulhat osztályozási feladat, ahol viszont mindenképpen érdemes lehet számolni ezzel a megoldással is, hiszen az egyik leghatékonyabb algoritmus az ilyen jellegű problémák körében. A RandomForest hatékony megoldás, így ez mindenképpen jobb módszernek tekinthető az adott probléma szempontjából. Ugyanakkor igazán kiemelkedő módon az XGBoost teljesített. Látható, hogy a hozzá kapcsolódó mérések során a tanulási idők átlaga a 0,1 másodpercet sem haladta meg.

Az osztályozási és regressziós idők a tanulási időkkel szinte azonos ábrázolású/nagyságrendi különbségű diagramot adnak, ám érdemes úgy szemlélni ezen információkat, hogy bármely algoritmus meglehetősen hatékonyan tette mindezt, hiszen a legrosszabb AdaBoost is csupán 0,3 másodpercnél kevesebb időt igényelt, igaz a teszt adatsorok száma, mindössze a tanítási adathalmaz méretének 46-oda volt. 32000 sorra tekintve ez körülbelül azt jelenti, hogy a predikciós idő mindössze a tanulási idő negyede volt a RandomForest, valamint az XGBoost esetén, továbbá az AdaBoost nagyságrendileg 1/6-a a tanítás során eltelt szekundumoknak. Ez azért fontos, mert azt a következtetést lehet ebből levonni, hogy egy jól betanított modell – aminek a tanítására fordított idő akár lehet hosszú is – később viszonylag gyorsan kiszámíthatja a jóslatokat, ha nem értelmetlenül nagy adathalmazon kerül futtatásra, hanem például bizonyos időközönként, így akár naponta kiértékelésre kerülnek az eredmények. Ez a jelenlegi használati eset („use-case") mellett, ettől elvonatkoztatva is legtöbbször elegendő kell, hogy legyen, mivel a begyűjtött adatokból leginkább utólagosan szükséges következtetéseket levonni, amik a jövőbeli döntéseket befolyásolhatják majd.

A következő diagramok az ismertetett predikciós algoritmusokat a hibák szempontjából mutatják be. Fontos megjegyezni, hogy az AdaBoost olyannyira nem tudott hatékonyan működni, hogy legjobb esetben is csak azt a szintet sikerült elérnie, amit azzal a becsléssel lehet, ha minden predikció eredménye a leggyakoribb értéket eredményezné.



22. ábra - Predikció: Min. average difference



23. ábra - Predikció: Min. MSE

Az átlagos eltérés, kiszámítása úgy történt, hogy a teszt adatsorokra adott volt az elvárt érték, és a kapott jóslat. A két érték eltérése került összeadásra minden egyes rekord esetén, majd ezeket az összes elem számával elosztva kapható az eredmény az aktuális végrehajtásra. A diagramon mind a három algoritmus esetén a lekisebb ilyen érték szerepel. Megfigyelhető, hogy az XGBoost teljesített a legjobban, ami a teljesen véletlenszerűen jóslatot sorsoló, azonos értékkészlettel (jelen esetben kb. 0-1600) dolgozó véletlenszám generátor eredményeihez képest több, mint 96%-kal teljesít jobban. Ha figyelembevételre kerül az az ismeret, hogy a nulla a leggyakoribb érték, és hogy ezért lehet, hogy érdemes minden sort nullára becsülni, akkor is valamivel több, mint 34%-os a javulás, ami szintén jelentős. A RandomForest is jól teljesített, és további előnye, hogy egy viszonylag könnyen megérthető algoritmusról van szó. A teljesen véletlen eredményekhez képest 95%-os az előrelépés, míg a leggyakoribb elemre történő becslés is 28%-os előrelépést hoz

Az MSE („Mean Squared Error") jelentése átlagos négyzetes eltérés. Kiszámítása a következő módon történik: $MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$, ahol „yi" az elvárt, a „kalapos" „yi" pedig a predikció értékét jelöli. (Az XGBoost kialakított prototípusa kiírja ezenkívül az átlagos négyzetes gyökeltérést – RMSE („Root Mean Squared Error") – is, a megkövetelt körökre bontva – ebből látszik, hogy az adott tanítási kör hozott-e javulás a tanítási és a tesztelési adathalmaz esetén.) A teljesen véletlen módon, adott értékkészlet tartományon belül generált számok MSE értéke nagyjából 790'000 körül alakult a tesztelés során.

Az XGBoost a négyzetes eltérés tekintetében több, mint 99%-os javulást hozott. Abban az esetben, ha a leggyakrabban előforduló érték kerül jóslásra az összes teszt sor esetén, akkor 53%-ot meghaladó javulás az eredmény. Ez jól mutatja, hogy az algoritmus nagyon hatékony is mindamellett, hogy gyors. A RandomForest is megfelelő módon teljesített. Szinte azonos százalékokat produkált, mint a legjobban teljesítő XGBoost, ami 99%-ot és 52%-ot jelent.

Az összehasonlított algoritmusok és eredményeik ismeretében, jelen problémakör esetén egyértelműen az XGBoost használata javasolt. Ez a legelőnyösebb mind a sebesség, mind a hibák szempontjából. Utóbbi aspektusból a RandomForest sem teljesít rosszul. Azt kell eldönteni, hogy egy komplex fejlődő megoldás, vagy egy könnyebben implementálható, egyedileg fejleszthető algoritmus használata a preferált adott esetben.

3 K-Means továbbfejlesztése

A megismert K-Means algoritmus továbbfejleszthető, hogy valóban az adott problémakörben elvárt csoportosítás legyen a kimenet, illetve, hogy a végeredményt gyorsabban állítsa elő, ami a „BigData” korában elengedhetetlen. A kimenet pontosítása különböző metaadatok felhasználásával, eltérő futtatási megfontolásokkal érhető el. A kiindulás során létrehozott klaszterek akár azonnal egy jó csoportosítást adhatnak, ezért fontos a jó inicializálás. Például az energetikai szektor esetén, ha ismert az adott háztartásban élők átlag életkora, az épület típusa, és így tovább, akkor ezek függvényében meg lehet adni az algoritmusnak egy kezdeti csoportosítást. Szintén létrehozható ismert fogyasztói átlaggörbék alapján egy lista, ami a kiindulásként választott centroidokat tartalmazza. Ez vélhetően jobb csoportosítást fog eredményezni, hiszen eredetileg a középpontok kiválasztása véletlenszerűen történik, míg ebben az esetben már optimális közeli centroidok szerepelhetnek kiinduló állapotban és így csupán kisebb korrigálás kell.

A kapott eredmény a kiinduló halmazok megfelelő megválasztásán kívül is javítható. Ki lehet próbálni több különböző „k”-val is a futtatást, és ezekből a legjobban kézenfekvőt kell választani. Ezt továbbfejlesztendő létezik hierarchikus klaszterszám változtatással apelláló megközelítés is. Az utóprocesszálás szintén egy lehetőség, vagyis a klasztereket lehet egyesíteni és szétválasztani az igényeknek megfelelően, egyéb információk ismeretében, hogy végül egy jobb csoportosításhoz jusson a felhasználó.

A sebesség fejlesztésekor a párhuzamosítás, az erőforrások horizontális, valamint vertikális skálázása juthat az ember eszébe. Ugyanakkor sok algoritmus, s így a K-Means is lehetővé teszi, hogy egyes lépései kicserélésre kerüljenek. Az ötlet tehát, hogy mi lenne, ha az egyre inkább fejlődő kvantum számítást fel lehetne használni bizonyos lépések felgyorsítására. Ebben a fejezetben kifejtésre kerül, hogy hogyan lehetséges ez, illetve maga a megoldás általános lesz olyan tekintetben, hogy az ötletet felhasználva szinte bármely mesterséges intelligencia algoritmus tartalmaz részeket, melyek kicserélhetők kvantum komputerrel hatékonyan végezhető feladatokra, ezzel javítva a teljesítményt.

3.1 Adatgenerálás

Az energetikai szektor napi szintű fogyasztási adatai átlagemberek számára nem érhetők el, továbbá a résztvevők esetleges adatgenerátorai sem hozzáférhetők. Az

algoritmus futása ugyan nem igényel többlet metainformációt a nyers adatokon kívül, elleben a K-Means kiértékeléséhez szükséges az az ismeret, hogy az adott pont melyik csoportba kellene, hogy tartozzon.

Ismert tény, hogy a tesztadatok értékét a mennyiségük mellett a minőségük adja. Jelen esetben tehát a cél az volt, hogy a generátor program képes legyen akár milliárdos nagyságrendű, valósághű rekordokat gyorsan előállítani, a következő metrikák figyelembevétele mellett. Fontos, hogy ne csak egy ország adatait lehessen legeneráltatni, mert akkor az eszköz terület függetlennek és mobilisabbnak tekinthető. Szintén szükséges, hogy az országok mellett más adatok is részletesen testreszabhatók, illetve bővíthetők legyenek. Éppen ezért érdemes a konfigurációt egy külön fájlban JSON, vagy épp XML formátumban tárolni. Ezzel lehetővé téve, hogy a program módosítása nélkül befolyásolni lehessen a generált adatokat.

A konfigurációnak figyelembe kell venni a fogyasztási mérések gyakoriságát, mivel a valóságban sem folytonosan, hanem bizonyos, diszkrét időközönként történik a mérőállás leolvasása (pl.: 15, vagy 30 percenként). A fogyasztásokat tekintve releváns adatokat úgy lehet generáltatni, ha megadható, hogy hány fogyasztóra vetítve szeretné a felhasználó a generálást, illetve melyek azok a tipikus felhasználói csoportok, akiket érdemes modellezni. Az is meghatározó, hogy a csoportok milyen arányban lesznek jelen.

A tipikus felhasználói csoportok rögzítése úgy történt, hogy mintavételezésre került az adott, eredeti csoport egy releváns részhalmaza, bizonyos, diszkrét időközönként, és ezen adatok átlagai eltárolásra kerültek egy számsorban, mint tipikus fogyasztási adatok. Vagyis 30 perces bontás esetén egy napra vetítve úgy adható meg egy görbe, ha 48 érték felvételre például „Wh”-ban (wattóra) megadva. Természetesen a felbontás is konfigurálható.

A fogyasztók szempontjából egyfajta alapvető bontás képezhető, a lakossági és vállalati ügyfelek megkülönböztetésével, viszont – ezeken a halmazokon belül – be lehet vezetni további alcsoportokat. A vállalatokat érdemes lehet méretük szerint legalább két csoportba osztani, így kialakíthatók, például kis-, és nagyvállalatok. Ezen belül szintén figyelembe vehető, ha egy vállalatban a termelés több műszakban történik, hiszen ez jelentős mértékben fogja befolyásolni a fogyasztást. A lakossági ügyfelek esetén többfajta halmaz kialakítására van szükség. Érdemes korosztály szerinti csoportokat kialakítani, hiszen a nyugdíjasok jellemzően napközben is otthon vannak, míg a gyermekkel nem rendelkező párok valószínűleg 8 és 17 óra között nem tartózkodnak otthon, így ennek

megfelelően náluk egy két púppal rendelkező görbe lehet a jellemző. A gyermekkel rendelkező családi fogyasztók biztosan magasabb értékeket produkálnak, mivel minél többen laknak egy adott háztartásban, jellemzően annál nagyobb a fogyasztás. Meg kell jeleníteni jelen kutatás egyik fő csoportját, a – megújuló energiát termelő – háztáji erőművel rendelkező tulajdonosokat is. Érdekes, illetve érdemes lehet figyelembe venni – ha hosszabb távú mérésekről van szó –, hogy az adott héten hétköznapokról, vagy hétvégékről van-e szó. Évszakonként is nagy különbségek lehetnek, például amikor nincs szükség fűtésre, és hosszabbak a nappalok, valamint a naposórák száma is magas.

Az adatgeneráló program Java nyelven készült. Konfigurációját JSON formátumú állományból olvassa, illetve van lehetőség az alapértelmezett konfiguráció alkalmazására, fájlba mentésére. A generált adatok CSV formátumban kerülnek mentésre, ahol az első sor az attribútumok neveit tartalmazza. Sorosítható a tipikus fogyasztási görbék adatai is, melyek segítséget nyújtottak a centroid optimalizációs mérések során.

Jelen applikációhoz a lakossági fogyasztási mérőhelyek számának kialakításakor a KSH [7] 2011-es, legutóbbi érvényes népszámlálási adatai kerültek felhasználásra. Az olvasható ki ebből, illetve abból a tényből, hogy a Magyarországon aktuális elektromosság felhasználási helyzetben a lakossági fogyasztók aránya közel 75%-os, hogy mindamellett, hogy megközelítőleg 3,6 millió lakossági fogyasztóval érdemes számolni, a vállalati, gazdasági felhasználók száma körülbelül 1,3 millióra tehető.

A felhasznált átlagos fogyasztások úgy kaphatók, hogy figyelembevételre kerültek a magyarországi háztartások éves fogyasztása, majd ez el lett osztva 365 nappal, 24 órával és 2-vel, mivel a program jelenleg 30 perces bontást kíván ábrázolni. Ennek megfelelően adott, hogy mi az átlagos fogyasztás egy háztartásban fél órára vetítve. Már csak árnyalni kell a helyzetet, vagyis ki kell alakítani a görbék lokális minimum, illetve maximum helyeit, hogy mikor fogyasztanak többet/kevesebbet az emberek a nap során. A lakossági átlag az NKM [8] oldalán található 2018-as évi adatok alapján került figyelembevételre, de természetesen pontosabb adatok birtokában a generáló sokkal jobban alaposabban, relevánsabban is konfigurálható.

A gazdasági fogyasztók esetén az aktuális alapértelmezett konfigurációs fájl azt feltételezi, hogy a gazdasági fogyasztók közül a legnagyobbak háromműszakos termelést végeznek. A jelenleg kialakított 5 fogyasztói csoport a következő: nyugdíjasok, családok, családok megújuló energia visszatáplálással, kisebb vállalkozások, nagyobb vállalkozások.

A konfigurációs fájl az alábbi módon épül fel:

```
{
  "numberOfDays": 1,
  "intervalsInADay": 48,
  "countries": [
    {
      "name": "hungary",
      "consumerCount": 4900,
      "consumerTypes": [
        {
          "name": "pensionerHousehold",
          "maxDifferenceFromAverage": 15,
          "percentageOfTheWholeConsumerCount": 27,
          "averageConsumption": [
            20,
            ...,
            125,
            20
          ]
        }, ...
        ]
      }, ...
    ]
  }, ...
    ]
}
```

24. ábra - Adatgeneráló konfiguráció felépítése

A „numberOfDays” mező segítségével lehet beállítani, hogy hány napnak megfelelő adat generálása a cél. Az „intervalsInADay” paraméter adja meg, hogy hány intervallumra lesznek bontva a napok. Jelenleg ez 48 értékű, ami 30 perces bontást ad, de például 72-re átírva 20 perces felbontást lehet elérni. Az „averageConsumption” szekció egy-egy tipikus görbét ír le. A „countries” lista tartalmazza a modellezni kívánt országokat. Ezen belül megadható az ország neve, a felhasználóinak összesített száma, majd ezt követik a definiált fogyasztói típusok. A fogyasztók számossága esetén nem probléma, ha például az adott ország 100-ad része kerül kiértékelésre, mert a fogyasztók esetén a „percentageOfTheWholeConsumerCount” mező százalékos arányt vesz figyelembe. A „maxDifferenceFromAverage” tulajdonság arra jó, hogy a programban működő véletlenszám generátort segít konfigurálni, s így olyan adatokat lehet előállítani, ahol valamilyen plusz-mínusz értékben – legfeljebb ezen mezőben meghatározott számmal – eltér az átlagostól az aktuálisan mért fogyasztás (a valóságosság végett).

A generált CSV tartalma az alábbi módon néz ki:

```
id,day,hour,minute,consumption
hungary_pensionerHousehold_0,0,0,0,34
hungary_pensionerHousehold_0,0,0,30,6
...
```

25. ábra - Adatgeneráló kimenete

A klaszterező algoritmusok teszteléskor szükséges a sorok megkülönböztetése, így az országnév, fogyasztói típus és egy inkrementálódó szám azonosítót alkot. Az ugyanazon id-val rendelkező sorok ugyanazon fogyasztóhoz tartozó, ám különböző időpontban vett mérési adatokat jelentenek. A nap, óra, perc attribútumok jelölik, hogy a generált napokból hányadikon és mikor történt a mérés. Az utolsó érték a fogyasztás, mely – a korábbiakban leírtak alapján – véletlen faktor figyelembevételével generálódik. Az applikáció jelenleg több, mint 1 milliárd rekordot képes előállítani ezen beállításokkal: 1 nap, 49000 fogyasztó (magyar fogyasztók 100-ada), 30 perces bontás.

3.2 Hierarchical Agglomerative Clustering

A hierarchikus klaszterezési megközelítések két csoportba sorolhatók. Az egyik az úgynevezett „bottom-up”, a másik a „top-down” irányt követi. Ez nem csak kifejezetten a K-Means esetén létezik, hiszen ezen megoldás könnyen implementálható bármely csoportosítást végző algoritmus használatakor.

Az „alulról-felfelé” építkező a lehető legkisebb (1 elemű) csoportokból indul ki, és minden egyes lépésben egyesítik a leginkább összetartozó klasztereket. Ezt a módszert szokás Hierarchical Agglomerative Clustering (HAC) algoritmus felépítésnek nevezni. A hierarchiát fa struktúra, vagy dendrogram (cluster-analízisben alkalmazott ábra, amely bármiféle: hasonlóságaik, ill. különbségeik alapján csoportosított, cluster-ekbe összevont objektumokat reprezentál) segítségével lehet ábrázolni. A fa gyökere a legnagyobb minden elemet tartalmazó klaszter, míg a levelek az egyesével vett adatvektorok.

A „felülről-lefelé” haladó megközelítés esetén az ellentétes irányból kell indulni. Egy nagy csoportot kell szétbontani úgy, hogy a kapott két kollekció az egymástól leginkább távol álló elemekből épüljön fel. Az így létrejött diszjunkt halmazokokra pedig ismételten végre lehet hajtani ezt a metódust, egészen addig, amíg végül egyetlen egy elem marad minden egyes csoportban. Az egyes lépések során kialakult új klaszterek mentén ki lehet választani, hogy az adott esetben melyik csoportosítás a legelőnyösebb (például az 5, vagy 10 egységet tartalmazó felosztás).

Az egyik nagy előnye a hierarchikus algoritmus futtatásnak, hogy a kilépési feltétel független lehet a csoportok számától, vagy éppen az adatvektorok távolságainak maximumától. Természetesen egyszerre csak az egyik teljesülhet az előzőek közül, ugyanakkor ilyen módon például a K-Means is könnyen használható úgy, hogy a csoportokban levő elemek maximális távolságát határozzuk meg.

Az igazi hátránya ezeknek az implementációknak a legalább $O(n^3)$ -ös komplexitás. Hiszen legfeljebb „n” darab csoport jöhet létre – mivel üres klaszterek létrehozása ilyen felosztások készítésekor indokolatlan. Tehát lényegében „n”-szer kell végrehajtani a kiválasztott algoritmust, úgy, hogy minden egyes végrehajtás az őt megelőző futás eredményét tudja felhasználni. Könnyen látható, hogy ilyen esetben a komplexitás a választott algoritmus összetettségétől is függ, hiszen maga ez a megközelítés csak a szemléletet, a módszert adja. Ugyanakkor naponta végrehajtott számítások esetén hasonló beérkező adathalmazokra érdemes lehet legalább az első alkalommal ilyen módon kiválasztani az adott alkalmazási körben leginkább alkalmas klaszterszámot, majd a későbbi felhasználások során már adott a lehetőség, hogy ismert „k”-val a K-Means algoritmus kerüljön futtatásra. Ebben az esetben csak egyszer kell a magas komplexitásból adódó hosszú futtatást végig várni, később már hatékonyan tudnak működni az egyszerűbb algoritmusok. (Esetleg bizonyos időközönként érdemes lehet a „k” újbóli meghatározása, a meglévő klaszterszám validálása végett, hogy a leghatékonyabb csoportszám legyen biztosan kiválasztva.)

A következő szakasz az algoritmus fő elemeit ismerteti.

Bemenet:

- egy adathalmaz, címkék, attribútum elnevezések figyelembevétele nélkül.

Lépések (bottom-up megközelítés):

1. Klaszter kialakítása minden egyes adatvektor számára.
2. A 2 klaszterből, melyek távolsága a legkisebb egymástól, közös csoportot kell létrehozni, így az eredetileg „N” darab halmazból „N-1” marad.
3. Amíg a kilépési feltétel nem teljesül ismételni kell az előző lépést.

Lépések (top-down megközelítés):

1. Egy nagy klaszter kialakítása, ami minden egyes adatvektort tartalmaz.
2. A leginkább elszeparált 2 heterogén kollekció készítése a kiinduló halmazból.
3. Minden egyes újonnan létrejött halmazra az előző lépés futtatása.

Kilépési feltétel:

- Addig iterál, míg minden adat különálló csoportba nem kerül (egyéb kilépési feltételek: max. iteráció szám, csoport szám, csoport elemeinek „közelsége”).

Kimenet:

- egy a kilépési feltételnek megfelelő csoportosítása a bemenetnek.

3.3 Előfeldolgozás

A klaszterezés – de leginkább a K-Means – esetén, a helyes beállítások megtalálása, leginkább a jól megválasztott kezdeti értékektől függ. Belátható, hogy a rossz kezdő értékek lassabb, de legtöbb esetben még pontatlanabb eredményt is adnak.

A K-means algoritmus alapból véletlenszerűen választja a kezdeti értékeit. Ez egy jó megoldás, hiszen teljesen ismeretlen környezetben is könnyen alkalmazhatóvá teszi a megoldást. Ugyanakkor, ha rendelkezésre állnak metainformációk az adatokról, akkor leginkább statisztikai eszközökkel lehet fejlődést elérni. Egyéb közismert adottságokat is ki lehet használni, mint például jelen esetben – az adatgenerálás során már kifejtett – általános fogyasztói csoportokat/görbéket, amik alapján a már meglévő, mesterséges intelligenciát használó rendszerek is működnek. Érdemes tehát megfigyelni, hogy milyen javulást lehet elérni ilyesfajta előfeldolgozással, illetve megvizsgálni, hogy ez hogyan érhető el pontos adatok hiányában.

A kutatási során elkészült egy adatgenerátor, ami az algoritmus tovább fejlesztéséhez szolgáltat nagyméretű bemeneti adatmennyiséget. Ezt az adatgenerátort viszont fel lehet használni arra is, hogy kezdeti konfigurációt adjon a K-Means futtatása során. Ezt az indokolja, hogy ezzel a metodikával elérhetővé válik az a tudás, hogy a legeslegpontosabb becslés során mekkora javulás érhető el adott környezetben.

A javulás mind a teljesítmény, mind a hatékonyság során tetten érhető, hiszen értelemszerűen a klaszterezési művelet korábban is megtalálhatja azt az állapotot, amikor a klaszterek már nem változnak, mint teljesen ismeretlen, véletlenszerű helyzetben. A kiértékelés során a több eredmény átlagaiból keletkezett értékeket reprezentálja a „26. ábra - Advanced K-Means: Elapsed time (s)” és a „27. ábra - Advanced K-Means: Percentage of correct elements and count of clusters”, melyek a 3.4 Kvantum számítás szekcióban kerülnek ismertetésre. Összegzésként elmondható, hogy az előzetes konfiguráció a véletlenül választott kezdőértékekkel szemben átlagosan 8-szor gyorsabban hajtódott végre, mely a korábban elért végállapot miatt lehetséges. Hatékonyság szempontjából a K-Means minden esetben felismerte, hogy a kiinduló csoportosítás megfelelő, így 100%-os pontosságot ért el az előfeldolgozott esetben, szemben az átlagosan 44%-ot megközelítő általános működéssel. Látható az is, hogy míg véletlenszerűen végrehajtott csoportosítás során nem mindig sikerült megtalálnia a programnak a helyes klaszterszámot, addig előfeldogozott esetben ez sikerült.

A korábbiakban tehát bebizonyosodott, hogy „megéri” előfeldolgozással foglalkozni. Az adatok eloszlását, lokális maximum, minimum helyeit figyelembe véve centroidok becsülhetők. Statisztikai eszközökkel – átlag, medián számítás stb. – elérhető, hogy bizonyos tulajdonságokból egy-egy releváns érték keletkezzen. Ezekből csoportok hozhatók létre például úgy, hogy mindig venni kell egy speciális értéket, majd a többi oszlopból egy-egy véletlenszerűen választott értékkel kiegészíthetjük az adott kezdő centroid hiányzó koordinátáit. Általános eljárást adni –, mely minden adathalmaz esetén tökéletesen működik – nem lehet, viszont ezen gondolatmenet mentén némi számítással, statisztikával megfelelő kiinduló középpontok választhatók.

A kutatás célját nem képezte, de mindenképp megemlítendő, hogy sok egyéb előfeldolgozási mechanizmust lehet alkalmazni. Nagy számokkal lassabban lehet műveleteket végezni, mint kisebbekkel, így célszerű lehet az adatokat normalizálni. A beolvasás során ügyelni kell a hiányzó értékekre, melyek helyett alapértelmezett számokat kell bevezetni. Az elkészült programban ez megtörténik, illetve ezen folyamat során az összetartozó sorok egy rekorddá alakulnak. Erre azért van szükség, mert az adatgenerátor kényelmes, átlátható módon úgy állítja elő a kimenetét, hogy a sorok időben különböző értékeket jelölnek, az adott azonosítójú fogyasztóhoz. Az adott azonosítójú elem több sorból felépülése a K-Means számára nem értelmezhető, így célszerű volt egységesen adott időrendben beolvasni a sorokat, majd az összetartozókból egy hosszú, de minden információt tartalmazó rekordot készíteni, azaz transzformálni a bemenetet. Természetesen – főleg más algoritmusok esetén – egyéb előkészítések is előfordulnak.

3.4 Kvantum számítás

A kvantum számítás – ahogy a bevezetésből is kiderült – az algoritmusok terén még gyerek cipőben jár, akárcsak maga a technológia elterjedtsége. Napjainkban nehezen, vagy csak igazán drágán juthat az ember egy kvantum működésű, vagy ahhoz közelítő eszközhöz. Ugyanakkor programokat fejleszteni elméletben, matematikailag is lehet. A gyakorlatiasság végett pedig használhatók ezen célra készített szimulátorok, melyek a kvantum számítógépektől a jövőben elvárt viselkedést reprezentálják, akkor is, ha ezek jelenleg nem képesek a hőn áhított teljesítmény javulást bemutatni. Ezzel ellenkezőleg ezen eszközök szimulációs erőforrásigénye jelenleg rosszabb futási időket eredményez – érthető okokból – a hétköznapi használatban levő számítógépeken.

A mesterséges intelligencia fejlődése impozáns képet mutat az elmúlt időszakban. Elmondható, hogy ezen terület megkerülhetetlen, sőt valószínűleg mindenütt használandó lesz a jövőben. Adódik a gondolat, hogy eme két technológiát érdemes lenne keresztezni egymással. Nincs lehetőség teljesen, minden aspektusból AI algoritmus kvantum számítógépre történő átültetésére a jelen ismeretek szerint, hiszen ezek az eszközök nem ezt a célt szolgálják. Ugyanakkor jelen fejezetben egy olyan megoldás kerül bemutatásra, ami nem csak a K-Means klaszterező algoritmus esetén, de egyéb más mesterséges intelligencia algoritmus esetén is felhasználható.

A K-Means algoritmus pusztán matematikai eszközöket használva képes, – mindenféle egyéb – metainformáció ismerete nélkül, csoportosítani a bemeneti adathalmazt. A működését vizsgálva adódik a lehetőség, hogy a sebességen javítani lehet, ha a vektorok közötti távolság mérést gyorsabban, vagy éppen adott feladathoz jobban illeszkedő metódusok végzik. A distancia számítás alapvető esetben Euklideszi módszerekkel történik, de a kvantum számítás is lehetőséget ad új algoritmus bevezetésére, bizonyos kritériumok teljesülése esetén.

Az új metodikára azért van szükség, mert például az Euklideszi távolság meghatározásra kvantum számítógépek esetén nincs lehetőség, hiszen az több qubitet igényelne, mint amennyit a technológia jelenlegi állását tekintve a közel jövőben meg lehet engedni, vagyis kvantum környezetben ez egy költséges feladat lenne. Nem feltétlenül szükséges azonban annak pontos ismerete, hogy milyen távol helyezkedik el a két vektor egymástól. Ehelyett elegendő egy alkalmas becslést találni a problémára, vagyis egy olyan metrikát, ami bizonyítottan arányos a rekordok közötti távolsággal. Szerencsére van ilyen kapcsolat, amit ki lehet használni, ez a vektorok skaláris szorzatához köthető.

A qubiteket a következőképpen szokták jelölni, hogy ne kerüljenek összetévesztésre a normál számokkal: $|0\rangle$, illetve $|1\rangle$. Fontos azt is megjegyezni, hogy a következő matematikai levezetés csak R^2-ben, azaz 2 dimenzióban működik, ezért a programban majd szükség lesz némi átalakításra, hogy az R^n-be tartozó vektorok mérete – ahol $n > 2$ – visszaszorításra kerüljön. Szintén fontos, hogy a 2 dimenziós vektorok normalizált módon kerüljenek felhasználásra, hiszen a qubitek értéke is a 0 és 1-es állapotok szuperpozíciójából adódik. A számítás során $|h\rangle$, és $|c\rangle$ qubit fogja reprezentálni rendre az adott helyvektort és a centroidot, míg az ezek állapotát röviden jelölje $|A\rangle$ ($|A\rangle = |h\rangle\,|c\rangle$). Ezen kívül szükséges lesz egy 0 kezdőállapotú segéd egységre is: $|0\rangle$.

A kvantum informatika az átlagos számítógépes áramkörökhöz hasonlóan kapukat alkalmaz. Ezek közül bizonyos kapuk ismerete szükséges a kialakítandó algoritmus megértéséhez. Ezeket az egységeket úgy lehet elképzelni, mint a háromdimenziós térben az egységgömbben a tengelyek mentén végzett forgatásokat. Matematikailag ezek mátrixokkal, továbbá komplex számok alkalmazásával reprezentálhatók.

A Hadamard kapu, egy qubit manipulálására alkalmas. Ez egy pi (180°) forgatásnak felel meg az „x”, illetve „z” tengely körül. Az egység mátrix megfeleltetése a következő:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

Az U3 kapu [9] szintén egy qubitet késztet változásra. Programozását figyelembe véve három paraméterrel rendelkezik ($\vartheta, \varphi, \gamma$), melyek segítségével ez az egység a 3 Euler-szöggel való forgatást valósítja meg, mely szögeket az attribútumok reprezentálják. (Az Euler-szögek írják le egy vektor tetszőleges térbeli helyzetbe állítását, melyhez három egymás után következő forgatás szükséges – a tengelyek mentén.) A programban az U3 kapura a qubitek kezdeti értékének megadása végett van szükség. Mátrix formában a következőképp írható fel:

$$U3(\vartheta, \varphi, \gamma) = \begin{pmatrix} \cos\left(\frac{\vartheta}{2}\right) & -e^{i\gamma}\sin\left(\frac{\vartheta}{2}\right) \\ e^{i\varphi}\sin\left(\frac{\vartheta}{2}\right) & e^{i(\varphi+\gamma)}\cos\left(\frac{\vartheta}{2}\right) \end{pmatrix}$$

A Fredkin kapu, másnéven „cswap”, vagyis irányított csere kapu arra hivatott, hogy három qubitet úgy manipuláljon, hogy az első qubitet változatlanul hagyja (ez lesz az irányító), továbbá amennyiben ennek az állapota 1, akkor a második 2 qubitet felcseréli. Mátrix alakban a következő módon írható fel (bal oldal), míg az igazságtáblája az alábbi (jobb oldal):

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\end{pmatrix}$$

Input			Output		
A	B	C	A’	B’	C’
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

A közelítő algoritmus a következő lépésekben levezetett [10] kapcsolatot használja fel, matematikai aspektusból.

$$d_{Euklideszi} = \sqrt{(c-h)^2}$$

Ahol „c” a centroid, „h” a helyvektor a kétdimenziós síkon. Ez az egyenlet átírható a következő alakúra ekvivalens módon.

$$d_{Euklideszi} = \sqrt{|c|^2 + |h|^2 - 2(c \times h)}$$

A $(c \times h)$ a skaláris szorzást jelöli. Viszont az is tudva levő, hogy normalizált vektorokról lévén szó a hosszuk és így azok négyzete is 1.

$$d_{Euklideszi} = \sqrt{1 + 1 - 2(c \times h)} = \sqrt{2 - 2(c \times h)}$$

Láthatóan sikerült egy olyan egyenlőségre jutni, ahol a csoportosítandó elem és a kiválasztott centroid, vektor reprezentációjának skaláris szorzata arányos az Euklideszi távolságukkal.

Az algoritmus oldaláról megközelítve továbbiakban levezetett egyenlet kapható. A kiinduló állapotban ($|0\rangle|\mathrm{A}\rangle, |\mathrm{A}\rangle = |\mathrm{h}\rangle|\mathrm{c}\rangle$, röviden: $|\mathrm{hc}\rangle$) a kezdetben 0-s állapotú segéd qubitre egy Hadamard kaput kell alkalmazni, hogy az szuperpozíciós állapotba kerüljön:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|\mathrm{A}\rangle$$

Ezt követően segéd qubitet egy irányított csere során a „cswap” kapu irányítójának tekintve az alábbi módon alakul a feltétel.

$$\tfrac{1}{\sqrt{2}}(|0\rangle|\mathrm{A}\rangle + |1\rangle|\mathrm{RA}\rangle), ahol\ |\mathrm{A}\rangle = |\mathrm{hc}\rangle, \text{míg}\ |\mathrm{RA}\rangle = |\mathrm{ch}\rangle$$

Ezen felépítés esetén a segéd qubitre egy újabb Hadamard kaput alkalmazva alakul ki a következő. Ahol „I” az identitás művelete, vagyis $|\mathrm{IA}\rangle = |\mathrm{A}\rangle$.

$$\frac{1}{2}(|0\rangle\,|(\mathrm{I}+\mathrm{R})\mathrm{A}\rangle + |1\rangle\,|(\mathrm{I}-\mathrm{R})\mathrm{A}\rangle)$$

Ebben a formában érdemes helyettesíteni $(\mathrm{I}+\mathrm{R})$-t, illetve az $(\mathrm{I}-\mathrm{R})$ rendre P_0, illetve P_1 változókkal, hogy szebb alakra jusson a képlet.

$$\frac{1}{2}(|0\rangle\,|\mathrm{P_0A}\rangle + |1\rangle\,|\mathrm{P_1A}\rangle)$$

Amennyiben ez a forma kerül vizsgálatra, akkor látható, hogy a kvantum számítás során alkalmazott qubit értékének becslése [11] esetén, annak a valószínűsége, hogy a segéd qubit 1-es állapotú, kifejezhető a fentiekből és a skaláris szorzás segítségével.

$$P(segéd = 1) = \frac{1}{2}\langle A|P_1|A\rangle = \frac{1}{2}\langle A|(I - R)|A\rangle = \frac{1}{2}(|A\rangle \times |(I - R)A\rangle)$$

Az $|A\rangle$ kvantum állapotba behelyettesítés, majd a zárójelezés felbontása után a következő egyenlőség áll fenn.

$$P(segéd = 1) = \frac{1}{2}(|hc\rangle \times (|hc\rangle - |ch\rangle)) = \frac{1}{2}(|hc\rangle \times |hc\rangle - |hc\rangle \times |ch\rangle)$$

Mivel $|hc\rangle \times |hc\rangle = 1$, továbbá $|hc\rangle \times |ch\rangle = (|h\rangle \times |c\rangle)^2$, ezért adódik a következő egyenlet:

$$P(segéd = 1) = \frac{1}{2} - \frac{(|h\rangle \times |c\rangle)^2}{2}$$

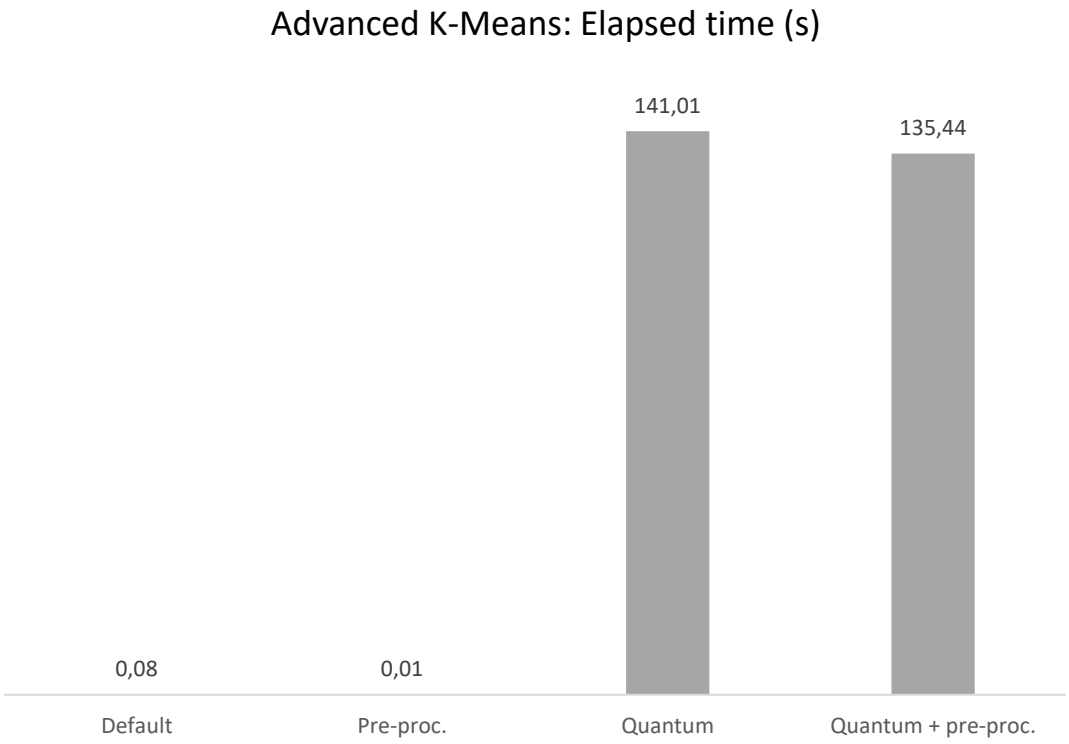
A matematikai egyenlet, illetve a kvantum algoritmus során is sikerült egy olyan egyenletet kapni, ami a vektorok skaláris szorzatával arányos. Természetesen ez nem teljesen pontos, hiszen nem áll fenn tökéletes arányosság a két egyenlet között (gyökös, illetve négyzetes), de alkalmas becslésnek bizonyult a gyakorlatban.

Az utolsó matematikai összefüggés, ami szükséges – K-Means program ily módon fejlesztett változatához – az U3 kapu kapcsán merül fel. Mivel kétdimenziósak a vektorok, így elegendő csupán ezen egység két bemeneti szögét (ϑ, φ) meghatározni, a harmadik tengely menti forgatás zérusnak tekinthető. Mivel a vektorok normalizáláson mennek keresztül, ezért [-1, 1] intervallumban kell, hogy elhelyezkedjen a koordinátáik értéke (jelen esetben ez tudva levő, hogy [0, 1] az intervallum, de így általánosabb a megoldás és felírható az alábbi összefüggés). Ennek ismeretében adódnak a következő egyenletek a szükséges szögek meghatározására. Jelentést tekintve „x” és „y” rendre az adott kétdimenziós vektorok első és második koordinátáját adják.

$$\vartheta = (x + 1) \times \frac{\pi}{2}, \qquad \varphi = (y + 1) \times \frac{\pi}{2}$$

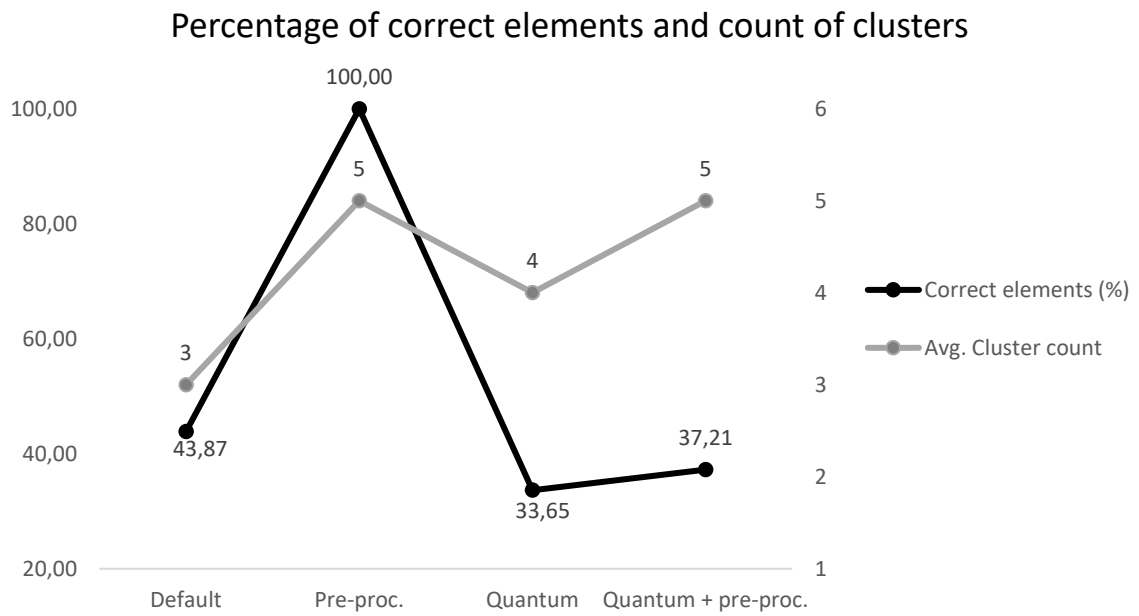
Az elkészült alkalmazás a Strange [12] nevezetű Java alapú szimulátort használja a kvantum számítások elvégzésére. Ez a könyvtár az egyik legelterjedtebb kvantum könyvtár a Java programozási nyelvet használók közül. Ennek ellenére akadnak hiányosságai, viszont könnyen bővíthető. A kutatás során ennek megfelelően új kapuk kerültek implementálásra (U3, CSwap). A jelenlegi verzióban található egyszerű végrehajtó szimulátor környezet (SimpleQuantumExecutionEnvironment) véletlenszerűen előforduló hibája miatt ezt ki kellett cserélni egy saját, javított változatra (TDKQuantumExecutionEnvironment), melynek során a kapcsolódó osztályok leszármazottal történő lecserélése is megtörtént a szükséges esetekben.

A korábbiakban említésre került, hogy jelen megoldás csak kétdimenziós vektorok kezelésére képes. Ennek megfelelően előfeldolgozásra van szükség, hogy a bemenetből két koordinátával rendelkező, azaz 2 attribútumos rekordok álljanak elő. Ezt úgy sikerült elérni, hogy az első koordináta a legmagasabb, míg a második az átlagos fogyasztási értéknek felel meg az adott napra (rekordra) vetítve. A normalizáláshoz a kvantum számítással működő távolság számító osztály rendelkezésére áll – az inicializálásától kezdődően – a teljes adathalmazra vett maximális fogyasztási érték, illetve a legmagasabb napi átlag. Azért, hogy a becslés pontosabb legyen, egy adott esetben 5-ször fut le a becslő. Ezen értékeket figyelembe véve kerül meghatározásra a legközelebbi centroid. A továbbiakban a kvantum számítást, vagy centroid optimalizációt, esetlegesen mind a kettőt, vagy egyiket sem használó végrehajtási esetek kerülnek bemutatásra.



26. ábra - Advanced K-Means: Elapsed time (s)

A kvantum működés szimulálása nehézkessé teszi a végrehajtáshoz szükséges idők releváns összehasonlítását, hiszen a diagram mutatja, hogy ezen esetben az átlagosan tized másodpercet is csak megközelítő futási idő a két percet is meghaladja. Ugyanakkor a centroid optimalizáció, vagyis a preprocesszálás során láthatóan gyorsabban állt be a végső állapot normál, illetve kvantum környezetet használó futtatás esetén is. Érthető a jelenség, hiszen a K-Means akkor fejezi be végrehajtását, ha nem változnak a csoportok, s amennyiben a halmazok állandósága korábban áll be, úgy a futás is rövidebb lesz.



27. ábra - Advanced K-Means: Percentage of correct elements and count of clusters

A fenti ábra reprezentálja – a már ismertetett négy esetben – az algoritmus által kialakított átlagos klaszter számot, valamint a helyes csoportba sorolt rekordok százalékos arányát. A klaszter számok érdekes módon kvantum környezetben jobban közelítették az elvárást, ami az újféle becslésnek a hozadéka. Az átlagosan helyesen besorolt elemek aránya úgy került meghatározásra, hogy az adott csoport leggyakrabban előforduló rekord típusa lett a csoport fajtája is. Ezt követően megszámlálásra került, hogy mi a jó elemek száma az összes klaszterbeli elemhez viszonyítva. Amennyiben ez minden halmazra kiszámításra került, akkor az összes elemmel elosztva a jó helyen szereplő rekordok sokaságát adódik, hogy mi a korrektül besorolt sorok száma. Szintén jó megoldás lehetne, viszont valamelyest más eredményt adna, ha a kialakult csoportok fajtája úgy kerülne meghatározásra, hogy 5 kötelezően különböző halmaz típust kellene kiosztani, az alapján, hogy mely elemfajták, mely csoportban foglalnak helyet legnagyobb számban. Ellenben ez a megoldás több mellék esetet tartalmaz, a típus definiálása során, így a korábban ismertetett egyértelműbb megoldás került felhasználásra az ábra elkészítésekor.

A diagramról leolvasható, hogy milyen hatékony a kezdeti centroid optimalizáció, mely mindkét esetben javulást hozott. Normál esetben ez több futtatást követően is tökéletes eredményt jelent. Láthatóan a kvantum számítás is működik, ugyanakkor rosszabb eredményekkel. Ennek oka egyértelműen a 2 dimenzióra történő váltásban, illetve a teljesen pontos távolság mérés és annak becslésének viszonylatában keresendő. Az algoritmust célszerű lenne több dimenziós esetekre is kiterjeszteni a későbbiekben.

4 Összefoglalás

Ebben a fejezetben a dolgozat során elkészültek továbbfejlesztési lehetőségeiről, valamint a kialakult konklúzióról lesz szó. (https://github.com/egytom/tdk2020)

4.1 Továbbfejlesztési lehetőségek

Az energetikai szektor számos olyan aspektust tartalmaz, amelyeket a továbbiakban fel lehetne térképezni, hogy azon területeken milyen algoritmusok lennének alkalmazhatók, így ez további kutatások alapját jelenti. Az elkészült megoldások közül a megfelelőket kiválasztva, egymásra épülő összetett rendszer létrehozása is egy jövőbeli cél lehet. Jelen tanulmányban elkészült adatgenerálót könnyen ki lehet egészíteni – a moduláris felépítéséből adódóan –, hogy például a gáz, víz és egyéb más területeken is alkalmazásra kerülhessen, s így további kutatási távlatokat lehetne elérni. A kvantumszámítás bemutatott megoldása, csak egy a sok lehetőség közül, így amennyiben megtalálásra kerülnek további feladatokhoz illeszkedő mesterséges intelligencia algoritmusok, akkor újabb kvantum metodikák megalkotása is lehetséges.

4.2 Összegzés

A kutatás során a cél továbbfejlesztett algoritmusok készítése és vizsgálata volt. A megvalósított alkalmazások hatékonyság növelése előfeldolgozással történt. Ennek során fogyasztói csoportok kialakítására két klaszterező, illetve napenergia hatásfok meghatározása végett három predikciós program lett fejlesztve. Az elkészített alkalmazások segítségével az algoritmusok valós felhasználási esetekben kerültek vizsgálatra. Analízisük során a végrehajtási idő, illetve a pontosság – predikció esetén átlagos és négyzetes eltérés, továbbfejlesztett klaszterezés esetén százalékos precizitás, illetve csoport szám – került figyelembevételre, diagrammokkal illusztrálásra.

A hatékonyabbnak ítélt klaszterező algoritmus továbbfejlesztése megtörtént. Ennek során elkészült egy újabb kiegészített alkalmazás, mely az alapvető vektorok közötti távolság mérés mellett, egy új kvantum számítással működő közelítő metodikát is reprezentál. A hatékonyság növelésének elérése érdekében további előfeldolgozási lépések kerültek bevezetésre, melynek során kifejtésre került a centroid optimalizáció, továbbá a hierarchikus végrehajtás lehetősége csoportosítás esetén. A tanulmányban ismertetve lettek az eredmények, illetve egyéb fejlesztési kiindulópontok.

Irodalomjegyzék

[1] M. Kubat, An Introduction to Machine Learning, Springer, 2017.

[2] V. K. Jaime Nava, Algorithmic Aspects of Analysis, Prediction, and Control in Science and Engineering, Springer, 2015.

[3] S. Akama, Elements of Quantum Computing: History, Theories and Engineering Applications, Springer, 2015.

[4] M. A. N. &. I. L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.

[5] L. Chen, „Boosting," [Online]. Available: https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725. [Hozzáférés dátuma: 2020].

[6] „XGBoost Kubernetes," [Online]. Available: https://xgboost.readthedocs.io/en/latest/tutorials/kubernetes.html. [Hozzáférés dátuma: 2020].

[7] „KSH," 2011. [Online]. Available: http://www.ksh.hu/docs/hun/xtabla/haztfogy/tablhf10_02_01.html.

[8] „NKM," 2018. [Online]. Available: https://www.nkmenergia.hu/aram/pages/aloldal.jsp?id=550565.

[9] „Qiskit documentation," Qiskit Development Team, 2020. [Online]. Available: https://qiskit.org/documentation/. [Hozzáférés dátuma: 2020].

[10] S. Anagolum, „K-Means," 2019. [Online]. Available: https://towardsdatascience.com/quantum-machine-learning-distance-estimation-for-k-means-clustering-26bccfbfcc76. [Hozzáférés dátuma: 2020].

[11] E. M. a. C. Singh, „Investigating and improving student understanding," Europe, 2017.

[12] „Strange," [Online]. Available: https://github.com/redfx-quantum/strange. [Hozzáférés dátuma: 2020].

[13] R. Kitchin, „What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets," 2016. [Online]. Available:

https://journals.sagepub.com/doi/full/10.1177/2053951716631130. [Hozzáférés dátuma: 2020].

[14] B. F. Joseph Feller, Understanding Open Source Software Development, London: Addison-Wesley, 2002.

[15] „GIT," Software Freedom Conservancy, [Online]. Available: https://git-scm.com/. [Hozzáférés dátuma: 2020].

[16] „IntelliJ IDEA," JetBrains, [Online]. Available: https://www.jetbrains.com/idea/. [Hozzáférés dátuma: 2020].

[17] „Java," Oracle, [Online]. Available: https://www.oracle.com/java/. [Hozzáférés dátuma: 2020].

[18] „Maven," Apache, [Online]. Available: https://maven.apache.org. [Hozzáférés dátuma: 2020].

[19] „Lombok," The Project Lombok Authors, [Online]. Available: https://projectlombok.org/. [Hozzáférés dátuma: 2020].

[20] „Hierarchical Agglomerative Clustering," [Online]. Available: https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/. [Hozzáférés dátuma: 2020].

[21] A. Dehghani, „Baeldung," [Online]. Available: https://www.baeldung.com/java-k-means-clustering-algorithm. [Hozzáférés dátuma: 2020].

[22] A. Saket, M. Sushil, T. Oncel és M. Peter, Semi-Supervised Kernel Mean Shift Clustering, IEEE, 2014.

[23] E. Raff, „JSAT," [Online]. Available: https://github.com/EdwardRaff/JSAT/wiki. [Hozzáférés dátuma: 2020].

[24] P. Grover, „XGBoost," [Online]. Available: https://towardsdatascience.com/clearing-air-around-boosting-28452bb63f9e. [Hozzáférés dátuma: 2020].

Ábrajegyzék

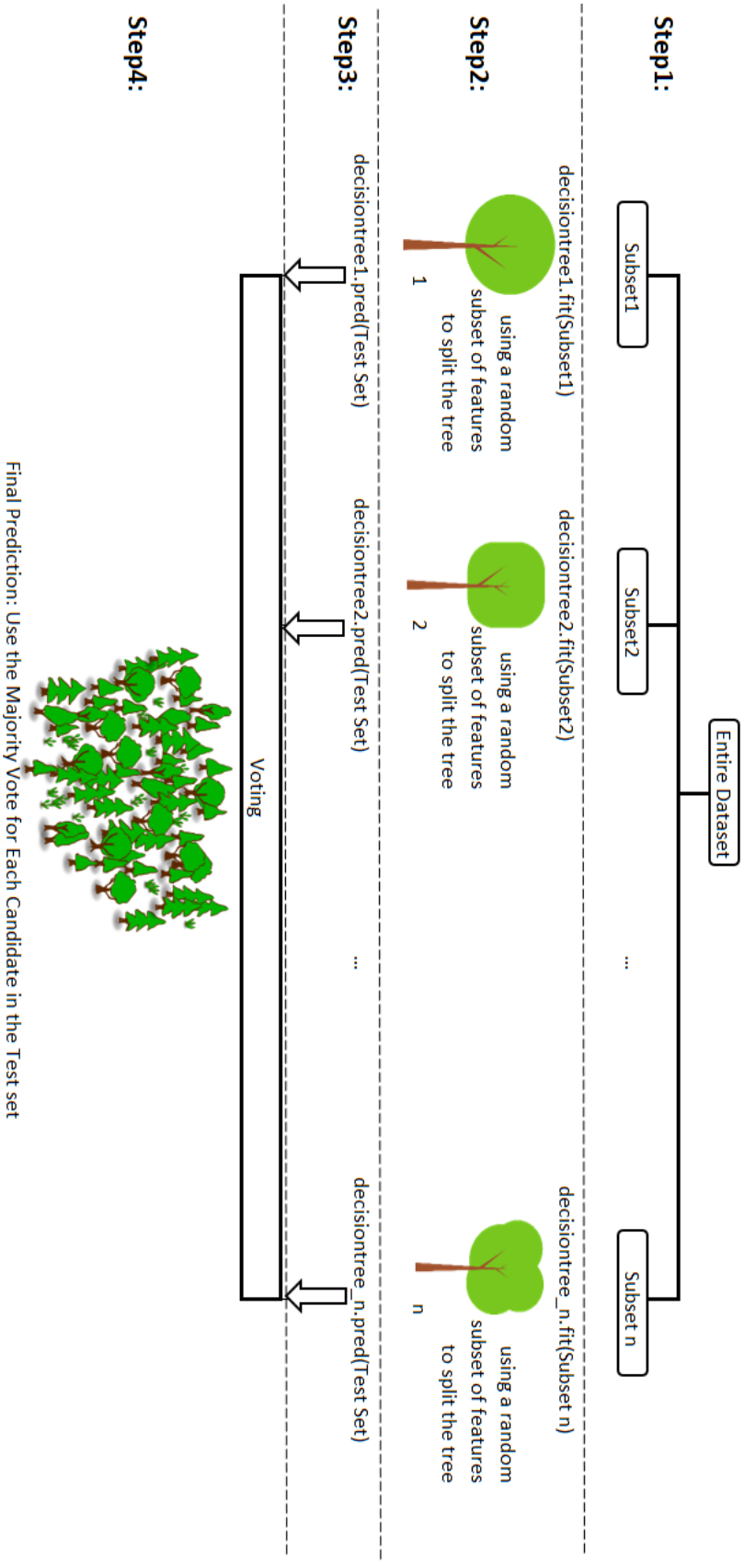
Melléklet

Fogalmak

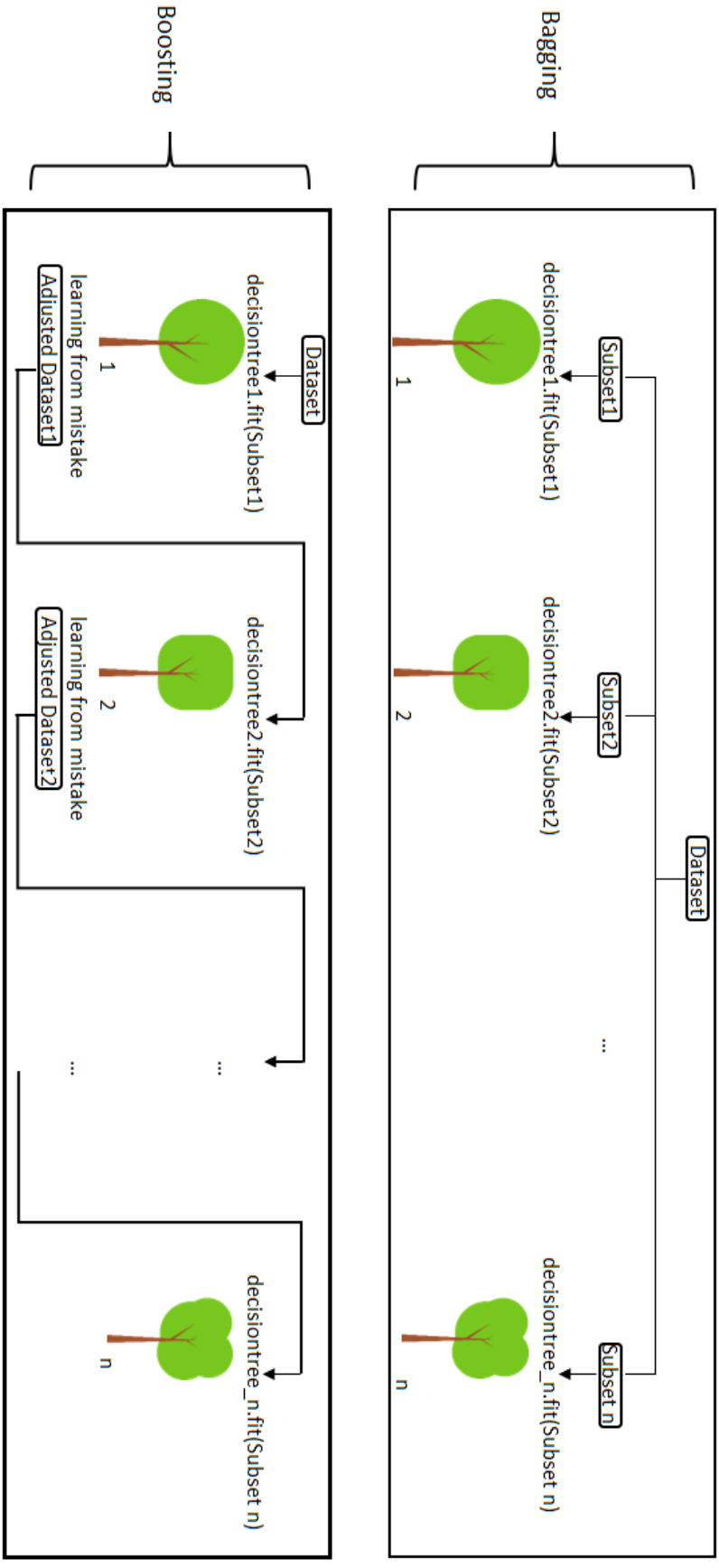
- **BigData:** hatalmas adatmennyiség, ami gyorsan hozzáférhető, változatos, kimerítő az adott témakör szempontjából, és skálázható. [13]
- **deep learning:** A mély tanulás a gépi tanulásra épít. Működése az emberi agyat veszi alapul, szokás úgy tekinteni rá, mint hálózatokra, hiszen az elnevezés arra utal, hogy egymásra épülő (rétegzett) lépésekből áll.
- **felügyelt tanulás:** a mesterséges intelligencia azon működése, amikor a tanulás folyamata irányított, a tanulási fázisban tudatva van az metódussal, hogy jó döntést hozott-e. Ezt szokás felügyelt környezetnek is nevezni. (Előnye, hogy képes adott környezetben javuló működést produkálni, nem csak a véletlentől függ a futása.)
- **felügyelet nélküli tanulás:** a mesterséges intelligencia azon működési módja, amikor a tanulás folyamata irányítatlan. Ekkor jellemzően nincs visszacsatolás, a megoldás elvégzi a feladatát, de nem tudja, hogy azt mennyire végezte jól. Ezt szokás felügyelt környezetnek is nevezni. (Előnye nem igényel előzetes ismeretet.)
- **klaszter:** lényegében csoportot jelent. A dolgozatban algoritmusok egy halmazát – melyek felügyelet nélküli környezetben működnek és alakítanak ki egységeket – hívják klaszterező algoritmusoknak.
- **kvantumbit (qubit):** kvantum informatikai rendszerekben, quantum számítógépekben a klasszikus számítógépek bitjeinek megfelelője az úgynevezett kvantumbit. Ezt röviden szokás qubitnek is nevezni, értéke nem csupán 1 és 0, hanem az egy és nulla különböző arányú szuperpozíciója, így végtelen sok eshetőség lehet. Méréskor a kalkulált bit értéket az aktuális qubit állapot adja.
- **machine learning:** A gépi tanulás a mesterséges intelligencia megoldások egy olyan alcsoportja, amely strukturált adatokkal dolgozó, a saját hatékonyságának fejlesztéséhez emberi beavatkozást nem igénylő módon éri el a kívánt eredményeket.

- **mesterséges intelligencia (AI, MI):** gyűjtő fogalom, magába foglal minden olyan algoritmust, megoldást, mely képes emberi beavatkozás nélkül intelligens döntéseket hozni. Intelligens döntésnek az tekinthető, ha a döntéshozó ismerete nélkül nem lehet megkülönböztetni, hogy azt ember, vagy gép hozta meg.
- **metainformáció/metaadat:** információ az adatokról. Olyan ismeret, mely segítségével könnyebb az adatokból jelentéssel bíró információt kreálni.
- **open source software:** nyílt forráskódú szoftver, melynek készítői a produktumot olyan licensszel látják el, mely lehetővé teszi a forráskód vizsgálatát, kiegészítését, módosítását a fejlesztői társadalom számára, valamint engedélyezett az így kiadott alkalmazás tetszőleges célokra történő felhasználása. [14]
- **pszeudokód:** egy fajta programkód, mely nem követ pontos nyelvi konvenciókat, sokkal inkább szabad nyelven írja le az adott algoritmus működését.
- **quantum számítás:** olyan számítás, melyet kvantum számítógépre írt programmal hajtanak végre. (Ez jelen esetben lehet szimulálás útján is, de végső soron az a cél, hogy magán a kvantum számítógépen történjenek meg ezek a lépések.)

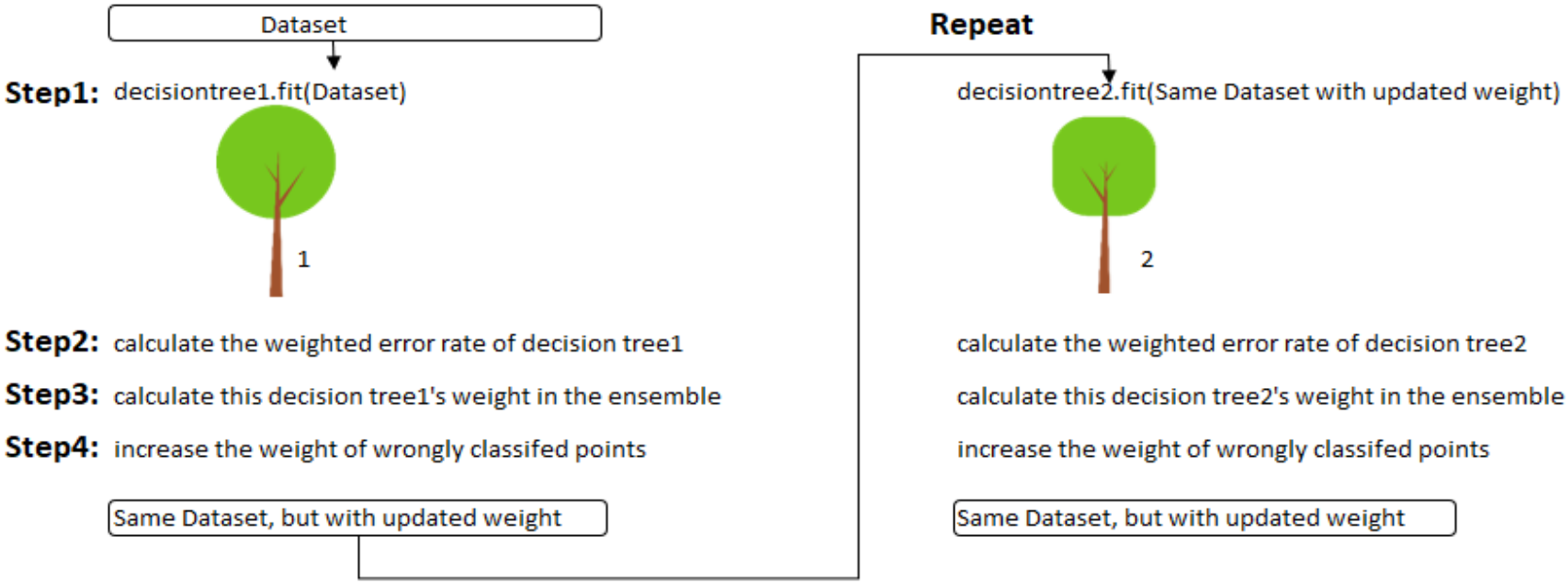
Ábrák



5. ábra - Random Forest algoritmus szemléltetése



9. ábra - Bagging és Boosting különbsége



10. ábra - AdaBoost tanítás szemléltetése

11. ábra - AdaBoost predikció szemléltetése

15. ábra - Gradient Boost tanítás szemléltetése

16. ábra - Gradient Boost predikció szemléltetése