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

Umlauf Zoltán

Adatfeldolgozási módszerek és megoldások kiberfizikai rendszerek proaktív karbantartásához

KONZULENS

Dr. Varga Pál

BUDAPEST, 2017

Tartalomjegyzék

Kivonat

Ipari környezetben az eszközök és eszköz-rendszerek folyamatosan nagy terhelésnek vannak kitéve. A költségek alacsonyan tartása miatt alapvető elvárásnak számít a rendszerek minél nagyobb rendelkezésre állása. Az eszközök karbantartása hagyományosan korrektív jellegű, vagyis a beavatkozás csak a meghibásodás után történik meg. Ezzel szemben egyre nagyobb teret nyer a proaktív karbantartási megközelítés, amelynek célja már a meghibásodás előtti beavatkozás. Ennek megvalósítása nem triviális, de megoldható már létező technológiákkal. A kiberfizikai rendszerek sokaságából álló hálózatok proaktív karbantartására a rendszer állapotát már most is figyelő szenzorok és beágyazott rendszerek szolgáltatásait kell összefogni egy lokális automatizálási felhő alapú megoldásban. Az ilyen felhőben teljesíthetők a valós-idejűségre és biztonságra vonatkozó követelmények, továbbá az adatok feldolgozására és elemzésére is könnyen fordítható nagy mennyiségű erőforrás. Egy így kialakított platform képes lehet a meghibásodások előre-jelzésére, valamint alkatrészek hátralévő élettartamának becslésére. Ennek segítségével a szükséges karbantartási és felújítási munkák jól ütemezhetők üzemidőn kívülre, így csökkentve a kiadásokat.

Az Európai Uniós MANTIS projekt [1] azt a nem kis feladatot tűzte ki célul, hogy az iparban megtalálható kiberfizikai rendszerekből álló rendszerek számára nyújtson egy platformot, amely proaktív karbantartási ökoszisztémát biztosít. Ennek általános megvalósításához rendkívül rugalmas és hatékony adatfeldolgozási módszertant kell létrehozni. Ehhez mindenképp szükség van nagy mennyiségű szenzor-adatra, jól alkalmazott gépi tanulási modellekre, valamint az adatok elő-feldolgozására.

A dolgozatban röviden bemutatom az ipari automatizálásban gyakran alkalmazott paradigmákat, majd kitérek a MANTIS projekt felépítésének és céljainak ismertetésére is. A dolgozat fő témája egy, a targoncák alkatrészeinek élettartam-becslését célzó, valós esettanulmány bemutatása, valamint az ehhez kapcsolódó adatfeldolgozási eljárások részletes ismertetése, amelyen dolgoztam. A dolgozat végén az elért eredményeket elemzem, és a további fejlesztési lehetőségeket részletezem.

Abstract

Industrial systems are under constant utilization, often in a demanding environment, therefore wear-outs have a likely occurrence. High availability is expected from these systems, while costs must still be kept low. Traditionally, the maintenance process happens in a corrective fashion, meaning that any kind of intervention only happens after the component (or the whole system) broke down. In contrast to this, proactive maintenance solutions are gaining popularity, which aim to have the intervention before the breakdown.

Realization of such a solution is non-trivial, but can be done with already existing technologies. Developing a proactive maintenance solution for a whole network consisting of Cyber-Physical Systems (CPS) can be done with the help of cheap commercial sensors and embedded systems, which are already in place to monitor the systems. The task then is to unite the data flows from these devices in a local automation cloud based solution. Such clouds should guarantee real-time processing while meeting strict, industrial grade security requirements, and provide the needed resources for processing and analyzing of the huge amount of inbound data. A platform like this should give failure forecasts and, for example, predict remaining useful life (RUL) for various equipment components. Based on this, maintenance related tasks can be properly scheduled then, which helps reducing costs.

MANTIS [1] is a European project which aims to solve this challenging task. It plans to provide a proactive maintenance platform for systems consisting of CPSs. To achieve this, we need to create a highly flexible and efficient data processing methodology. Great amount of sensory data, good data preprocessing and well-applied machine learning algorithms are necessary for achieving this goal.

In this paper, I give a brief overview of the most used paradigms in industrial automation and then discuss the structure and goals of the MANTIS project. The main topic of this paper is to present an industrial forklift-related use case for condition based maintenance (CBM). In here, I describe in detail my proceedings in the offline data processing phase. Finally, I analyze the achieved results and lay out future work.

1 Bevezetés

Már évek óta jellemző trend, hogy az emberekhez hasonlóan az eszközök széles skálája is kapcsolódjon az Internetre. A Gartner aktuális előrejelzése szerint 2017 végére 8,4 milliárd Internetre kapcsolt IoT eszköz lesz, számuk 2020-ra pedig várhatóan eléri a 20,4 milliárdot [2]. Ez és az ehhez hasonló előrejelzések egy automatizált jövőt vízionálnak, ahol az egyes kiberfizikai rendszerek (CPS – Cyber-Physical Systems [3]) globális szinten össze vannak kapcsolódva, és képesek együtt működni.

Az automatizálásra a lehetőséget a hálózatok végpontjain elhelyezett szenzorok és beágyazott eszközök teremtik meg. A szenzorok és beavatkozók segítségével a CPS állapota nyomon követhető, ha képesek vagyunk a szenzorok által generált nagy mennyiségű adatot hatékonyan feldolgozni. A kiberfizikai rendszer tehát annyival tud többet például egy hagyományos ipari rendszernél, hogy a begyűjtött adatok segítségével aktuálisan nyomon követhetők a rendszer használati jellemzői, állapotváltozásai, és egyéb fizikai jellemzői is – és megfelelő modellezéssel előre jelezhetők a fizikai rendszerek állapotváltozásai is. A szenzorok adatai különböző céllal működő feldolgozó egységekhez kerülnek, végül pedig egy megjelenítő eszközön (HMI – Human Machine Interface) keresztül a rendszer operátorához. Az adatfeldolgozás célja sokrétű lehet: megismerni a szenzorok által mért környezeti hatások korrelációját, mérni a vizsgált rendszer elhasználódását (RUL – Remaining Useful Life), vagy például a meghibásodás okának meghatározása (RCA – Root Cause Analysis).

Ezekkel a módszerekkel kialakíthatók proaktív felügyeleti és karbantartási stratégiák [4] (CBM – Condition-based Maintenance). A proaktív karbantartás lényege, hogy egy eszköz paramétereinek folyamatos megfigyelésével becsülni tudjuk annak elhasználódását (RUL érték meghatározása), amely segítségével még a meghibásodás előtt megrendeljük a szükséges csere-alkatrészeket, és ütemezzük a karbantartást egy megfelelő időpontra, hogy az eszköz működési ideje maximalizálva, a tervezetlen leállási idő pedig minimalizálva legyen. Ez a megközelítés rengeteg pénzt képes spórolni a hagyományos megközelítéshez képest (karbantartás ütemezése, amikor már megtörtént a meghibásodás) ipari rendszerekben, hiszen jelentősen lerövidíti az eszközök leállási idejét, valamint nem szükséges csere alkatrészek hosszú távú tárolásáról sem gondoskodni.

1.1 Jellemző paradigmák és technológiák

Kiberfizikai rendszereket számos, erősen különböző felhasználási területen alkalmazhatunk (pl. járművek, ipari rendszerek, energiahálózat), így nem meglepő, hogy a használt módszertanok és technológiák magas fokú heterogenitást mutatnak. Ezek közül most kettőt szeretnék kiemelni, amelyek jelen dolgozat szempontjából kulcsfontosságúak.

1.1.1 Cloud és edge computing

A több kiberfizikai rendszerből álló rendszerek (CPSoS – Cyber Physical Systems-of-Systems) architektúrája egy erősen kutatott terület, melynek célja egy olyan megoldás meghatározása, amely a CPS-ek között biztosít magas fokú együttműködési lehetőséget. A már meglévő cloud technológiák képesek ennek az alapjait biztosítani az alábbi szolgáltatásokkal:

1. „On-demand” számítási és tárolási erőforrások használata, amelyek bárhonnan elérhetőek. Ez elengedhetetlen a nagy mennyiségben előálló, korrelálható adattömeg (Big Data) hatékony feldolgozásához.
2. Kommunikációs lehetőségek biztosítása a különböző cloud szolgáltatások között. Egy ilyen rugalmas infrastruktúrát használva az egyes CPS-ek igénybe vehetik más CPS-ek szolgáltatásait és funkcionalitásait az interneten keresztül.

A hagyományos cloud szolgáltatások hiába erősen elosztott jellegűek, modelljük az adatforrás szempontjából még mindig erősen centralizált. Emiatt ezek a rendszerek nem használhatóak igazán hatékonyan az IoT világában. Erre reagálva született meg a fog/edge computing architektúrája, amely célja a cloud szolgáltatások közelebb vitele az adatok forrásához, a hálózat „szélére”. Ebben a modellben az adatokat szolgáltató fizikai eszközök és a cloud szolgáltatásokat nyújtó (kisebb méretű) adatközpontok között a késleltetés lecsökken, amely bizonyos alkalmazásoknál kritikus.

1.1.2 Service Oriented Architecture

A Service Oriented Architecture (SOA) paradigma gyorsan egy bevett megoldás lett információ küldésére és fogadására az internet világában. A SOA modell céljai a hatékonyság, együttműködés és az agilitás fokozása a vállalati környezetben. Lényegében a SOA egy olyan megközelítése a hálózati architektúrának, ahol a hálózat szereplőinek

flexibilitása egy főszempont, mivel magukat a szolgáltatásokat tekinti a rendszer építő-elemeinek. Nincsen szabványosítva sem az, hogy milyen tulajdonságokkal kell rendelkeznie egy szolgáltatásnak, sem azok implementálásának a módja, ezzel szemben az alkalmazások üzleti logikája a szolgáltatásokon keresztül reprezentáltak [5].

A webszolgáltatások [6] használata az iparban segített elterjeszteni a SOA elvek alkalmazását. Manapság számos technológia és szabvány használható webszolgáltatások fejlesztésére szoftver architektúrák számára, de a ténylegesen használt szabványoktól függetlenül is a webszolgáltatások két csoportba sorolhatók: Simple Object Access Protocol (SOAP) és Representational State Transfer (REST). Mindkét megoldás arra ad választ, hogy hogyan férjünk hozzá webszolgáltatásokhoz, és megvannak a maguk előnyei. Ezekre az elvekre alapozva számos SOA alapú megoldás született már, amelyek a különböző hálózati elemek integrációját/együttműködését hivatottak megvalósítani. Ilyen például az Arrowhead nevű EU projekt is, amelyen az önálló laboratóriumom keretén belül dolgoztam.

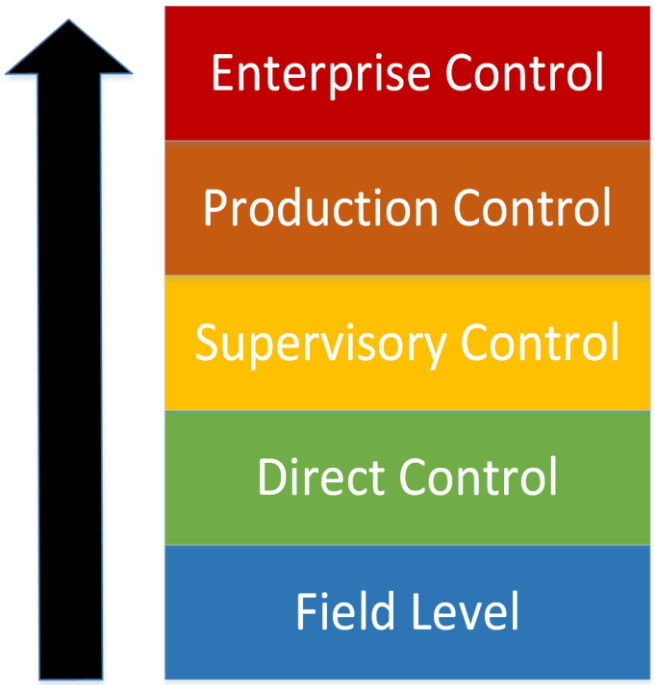
1.2 Ipari automatizálás szintjei

Az ipari automatizáltság egyre hatékonyabb megoldásokat tesz lehetővé, amelyben újonnan a CPS-ek elterjedése is szerepet játszik. Most már szinte minden olyan környezetben megtalálhatóak a beágyazott eszközök, ahol ipari gépezetek is találhatóak. A begyűjtött adatokat felhasználva Big Data algoritmusok számos területen képesek analízist és predikciót végezni, amely segítségével eszközök és rendszerek karbantartására új lehetőségek adódnak.

Az ipari automatizáláshoz köthető folyamatokat gyakran az ISA-95-ös szabvány [7] szerint sorolják különböző szintekbe. Az egyes használati eseteknél kis különbségek lehetnek a szintek formalizmusában, de a szintek egy általános definícióját az 1.1. ábra mutatja:

- 0. szint: ez a szint tartalmazza a szenzorokat és beavatkozókat, amelyek közvetlenül kapcsolatban vannak a folyamattal vagy a gépezettel.
- 1. szint: a beágyazott eszközök szintje, ahol PLC-k, jelfeldolgozó egységek vagy ipari PC-k végzik a 0. szint eszközeinek vezérlését.
- 2. szint: az ipari folyamat felügyelő szintje, ahol az operátorok nyomon tudják követni a gépek állapotát, valamint tudnak módosítani a folyamaton.

- 3. szint: a gyártási folyamat egészét foglalja magában, tehát része a karbantartás, gyártás, minőség-ellenőrzés és a készlet menedzsment.
- 4. szint: többnyire menedzsment funkciókból áll, a gyártási folyamat céljának elérését segíti az egyes lépések ütemezésével.



1.1. ábra: Ipari automatizálás szintjei [8]

A 3. szinttől kezdve a folyamat már az eszközöket összekapcsoló hálózatot használva működik. Itt már a SOA koncepciónak megfelelően az adatok szolgáltatások segítségével érnek célba, ezáltal rugalmas és jól skálázható rendszereket lehet elérni.

1.3 Fennálló nehézségek

Bizonyos esetekben a hálózatok szélén keletkező szenzor adatoknak sokat kell „utazniuk”, míg több feldolgozó egységen is áthaladva az információ kinyerésre kerül, és hasznosítják. Ezt a folyamatot rengeteg különböző típusú kommunikációs protokoll és architektúra segítheti, amelyek együttműködése magas-szintű problémákat vet fel. Ilyen például az adat szemantikája, az üzenetek struktúrájának problémája, azaz hogyan találunk közös nyelvet, amit minden fél képes feldolgozni.

Sok esetben a szenzorok adatai elegendően nagy idő- és értékfelbontásban rendelkezésre állnak, de kommunikációs korlátok ennek átvitelét a hálózaton megakadályozhatják. Ilyenkor az adatok alacsony szintű előfeldolgozása/szűrése

szükséges, amely magában foglalhatja az adatok időszakos tárolását is. Fontos, hogy a szűrt adatok még mindig jól reprezentálják a megfigyelt fizikai folyamat jellegét, amely alapján a felhőbeli feldolgozó egységek eldönthetik, hogy szükség van-e az adatok szűrésmentes verziójára is.

Ebből következik, hogy a hagyományos kliens-szerver kapcsolatok helyett itt olyan megoldásra van szükség, ahol a kapcsolatok kétirányúak és perzisztensek. Ilyen fajta kommunikációt képesek megvalósítani a platform független Message Oriented Middleware (MOM) protokollok, amelynek az egyik *lightweight* implementációja az IoT világában közismert MQTT [9] (Message Queue Telemetry Transport). Ez az üzenetküldési protokoll publish/subscribe jellegű, vagyis az adatforrások a bróker megfelelő virtuális csatornájára (topic) küldi az adatait, majd a bróker aszinkron módon továbbítja ezt mindazon felek számára, akik az adott csatornára feliratkoztak. A megoldás fő előnye az, hogy jól használható sávszélesség korlátozott helyzetekben, mivel az üzenetek minimális overheaddel rendelkeznek. Emellett a brókerek képesek teherelosztásra is egymás között. A brókereknek nem kötelessége tárolni az üzeneteket, de általában a csatornák legutolsó üzenetét eltárolják, amit az új feliratkozó felek így azonnal megkaphatnak.

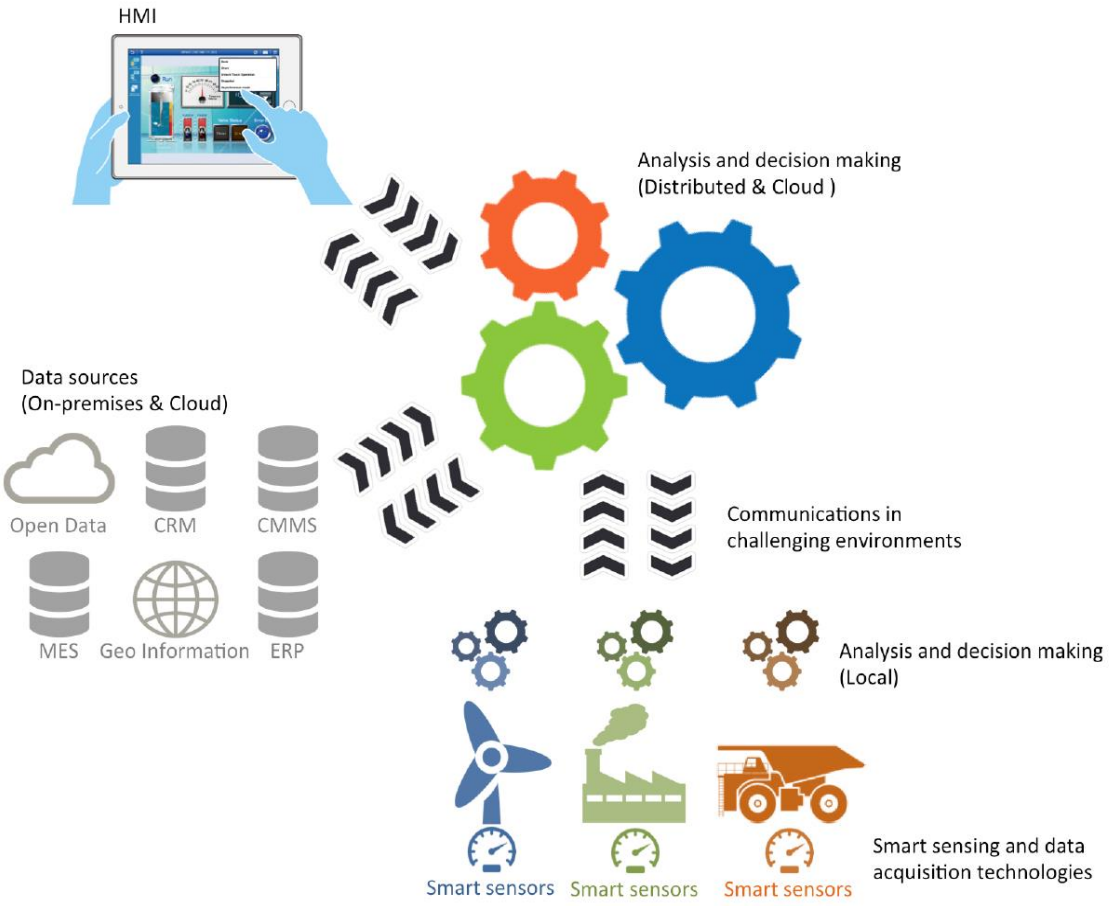
A MOM alkalmazása viszont nem oldja meg azt a problémát, hogy a különböző származású rendszerek adat reprezentációja és üzenet struktúrája teljesen eltérő lehet. Ezen a problémán hivatottak segíteni a MIMOSA [10] szabványok adat modelljei ipari alkalmazásokhoz. A MIMOSA adat modellje rendkívül általános és részletes ahhoz, hogy az ipari alkalmazások minden doménjében használható legyen, de sajnos emiatt erőforrás korlátozott eszközöknél nem célszerű az alkalmazása. Ilyenkor érdemes a modellnek csak a használati esethez releváns részeit használni, megőrizve a kompatibilitást a teljes MIMOSA modellel.

A gépi tanuláshoz és statisztikai modellek megfelelő betanításához nagyon nagy mennyiségű adat szükséges a múltból (historical data). Emellett a valós idejű adatok aggregálása a rengeteg forrásból is nagy kihívás lehet, a CBM (condition-based maintenance) hatékony megvalósítása pedig magában foglalhat számos funkciót (a RUL online nyomon követése, meghibásodások detektálása és analizálása, hatékony ember-gép interfész (HMI) stb.). Ezen funkciók megvalósításához elengedhetetlen a Big Data keretrendszerek használata, amelyből a legkényelmesebb megoldások a hardver erőforrásokat is biztosítják (pl. Microsoft Azure). Azonban, ha a szenzorok (szinte)

közvetlenül küldik be az adatokat a távoli harmadik fél által kezelt felhőbe, az biztonsági kockázatokkal járhat. Ilyenkor az adatfeldolgozást több szintre szükséges bontani, ahol az előfeldolgozást egy zárt hálózaton céleszközök végzik. A helyi zárt hálózat és a távoli felhő között emiatt is folyamatos kommunikáció szükséges.

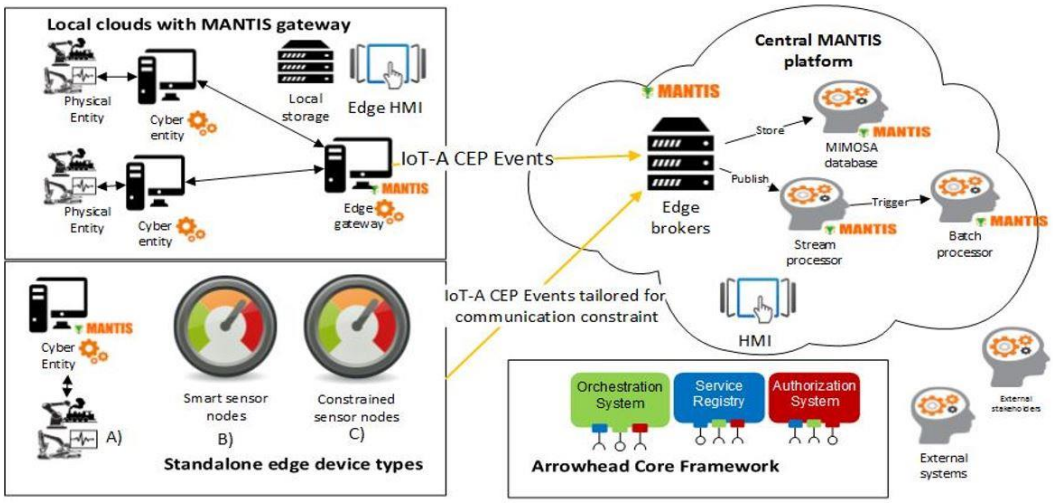
2 A MANTIS projekt

A MANTIS projekt [1] egy kiterjedt, kutatás-fejlesztési célú konzorcium munkája, amelyet az Európai Unió támogat. A projekt célja egy olyan platform kialakítása CPS-ekből álló rendszerek, amely már létező technológiákat alkalmazva biztosít proaktív karbantartási ökoszisztémát iparágak minél szélesebb skálájának (energia termelés, gyártási folyamatok, szállítás). Ennek megvalósításához a platform felhasználja a mindenütt jelenlévő szenzorok és beágyazott eszközök képességeit, valamint az elmúlt évtizedben kifejlett felhő inftrastruktúrák által kínált hatalmas mennyiségű on-demand számítási és tárolási erőforrásokat. A cél megvalósításához a platformnak számos ipari technológiával kell együttműködnie, amely technológiai és szervezési nehézségeket is felvet. A szervezési nehézségek enyhítése érdekében a platform kulcsösszetevője a middleware kommunikációs réteg, amelyhez új modulok csatolhatóak anélkül, hogy a meglévő rendszereken változtatni kellene [11].



2.1. ábra: A MANTIS általános koncepciója [1]

Ahogyan az a 2.1. ábrán is látható, az adatfeldolgozás két szinten történik az általános koncepció szerint. A helyi, alacsony szintű feldolgozás egyszerű jelek kinyerésével és analizálásával foglalkozik, a kiválasztott fizikai erőforrás viselkedésének megértése érdekében. Feladata a szenzor adatok aggregálása, a zajok és hibás adatok kiszűrése. Ehhez olyan metódusokat használ, mint például Kálmán-szűrő vagy a fuzzy logika (elmosódott halmazok logikája). Az alacsony szintű feldolgozás eredményeként adódó komplex adatok begyűjtésével és analizálásával foglalkozik a központi, elosztott jellegű feldolgozó egység. Célja a fizikai erőforrások viselkedésének globális megértése, amelynek eredményeit az ember-gép interfészen (HMI – Human Machine Interface) keresztül juttatja el az operátorokhoz. Olyan adatbányászati módszereket lehet itt használni, mint például adatok klaszterbe osztályozása, korrelációk keresése, *outlier* bejegyzések azonosítása, regressziós analízis.



2.2. ábra: A MANTIS platform alkotóelemei [12]

A 2.2. ábra a MANTIS platform általános architektúráját mutatja be. A felhő alapú platformhoz a gateway szervereken keresztül (edge brokers) tudnak csatlakozni eszközök, amelyekből 4 fajtát különböztetünk meg.

Ha rendelkezésre álló erőforrások és képességek szerint rendezzük őket sorba, akkor az első típusú eszközök a korlátozott képességű szenzorok. Az eszközök ezen csoportjának számítási kapacitása és kommunikációs képességei szűkösek, kommunikációra valamilyen vezetéknélküli protokollt használnak. Ezek a szenzorok gyakran rendelkeznek egy alvó állapottal energia megtakarítás céljából. Ilyenkor az eszköz periodikusan vált az aktív és alvó állapota között, idejének nagy részét az utóbbi állapotban töltve; így sokáig működő képes tud maradni. Aktív állapotában elküldi a mért

mennyiségét a platform számára, a lehető legrövidebb üzeneteket használva. A platform rendelkezik egy külön interfésszel ezen eszközök támogatására.

A második típusba az intelligens szenzorok tartoznak, amelyek feldolgozási kapacitása jelentősen nagyobb, valamint rendelkezhetnek adatok tárolására szolgáló memóriával. Ezek a szenzorok már képesek egyszerű előfeldolgozásra, így a platform a kétoldalú kommunikáció során kérhet a szenzortól folyamatos küldést és előfeldolgozott, felhalmozott küldési formát is.

A harmadik típusba már komplett CPS-ek tartoznak, amelyek maguk is képesek analizálni már a mért adatokat, szenzorokhoz képest nem rendelkeznek semmilyen korlátoltsággal.

Az utolsó típus, amit a platform megkülönböztet, az a CPS-ek csoportjából álló lokális felhők, amelyet valamilyen szempont miatt érdemes egy logikai entitásként kezelni a platform szempontjából.

Maga a MANTIS platform is 4 fő részből áll, ahogyan azt a 2.2. ábra is szemlélteti. Ezek a gateway szerverek, a stream feldolgozó egységek, a központi adatbázis, valamint a batch feldolgozó egységek. A gateway szerverek célja a bejövő forgalom begyűjtése és aggregálása. Implementálása során úgy kell kialakítani, hogy jól skálázható legyen ez az egység. A kimenete a megfelelő átalakítások után a központi MIMOSA adatbázisba, és a stream feldolgozó egység felé megy. A stream feldolgozó egység feladata a folyamatosan beérkező adatok alapján bizonyos események predikciója, érzékelése, valamint a megfelelő személyek értesítése a HMI-n keresztül. Jelenlegi állás szerint ez az egység a Storm keretrendszer segítségével lesz megvalósítva, és képes lesz eszközök RUL értékének számontartására, meghibásodások előre jelzésére, valamint váratlan meghibásodás esetén RCA kezdeményezésére a batch feldolgozó egység értesítésével. A batch feldolgozó egység feladata a komplexebb algoritmusok futtatása, amelyek nem kivitelezhetők valós időben. Ilyen például a különböző gépi tanulások képzési fázisa, amivel RUL modelleket lehet felállítani. Szintén ennek az egységnek a feladata az RCA elvégzése, amelyhez lekérheti a korábbi meghibásodások körülményeit az adatbázistól. Külső rendszerektől is gyűjthet adatokat, hogy megfelelő döntést tudjon hozni a karbantartáshoz.

A Service Oriented Architecture tervezési mintának megfelelően az előbb említett modulok interfészein megvalósuló adatcserére tekinthetünk úgy, mint szolgáltatásokra.

Ebben a szemléletben használható az Arrowhead keretrendszer [13] a MANTIS platformmal közösen: ekkor az Arrowhead a modulok közti együttműködést vezérli. Az Arrowhead képes dinamikus módon, futási időben megmondani az egyes moduloknak és a platformhoz csatlakozó eszközöknek, hogy hová kell csatlakozniuk adott szolgáltatás eléréséhez [14].

3 Adatfeldolgozás

A továbbiakban a MANTIS projekt keretében végzett adatbányászati munkámat mutatom be. Az egyetemi csapattal a német STILL működik együtt, mint ipari partner, aki a használati esetet és az adatokat biztosítja. Ez a németországi központú nemzetközi vállalat raktári felszerelések és eszközök gyártásával foglalkozik, fő fókusza pedig az elektromos, dízel és gázüzemű targoncák gyártása.

3.1 A felhasználási eset

A STILL világszerte számos vállalat számára biztosít targoncákat bérlés formájában, amelyhez folyamatos szervízelési rendelkezésre állás is társul, hogy a targoncák minél több időt töltsenek üzemképes állapotban. A STILL számára természetesen a szervízelés olcsóbb lenne, ha a megelőzhető meghibásodásokról előre tudnának, és preventitív módon beavatkozhatnának. A targoncák vezetőitől sajnos nem számíthatnak visszajelzésekre semmilyen formában a gyakorlatban, ezért az egyetlen lehetőség a begyűjtött adatok elemzésében rejlik. Az adatok begyűjtése egy GSM modem segítségével történik, amely hozzáfér a nem kritikus információkat tartalmazó CAN busz adatokhoz.

A STILL munkatársaival hosszas egyeztetések során esett arra a választás, hogy kezdetben a rendelkezésre álló adatok segítségével a kerekek elkopását próbáljuk előre jelezni, vagyis egy RUL-modell felállítása a cél a gumikra vonatkozóan. Sajnos a feladat korán sem triviális, hiszen a kerekek vastagságának mérésére nincs a targoncákon céleszköz. Ha lenne is ilyen szenzor, a gumi kopás jellegéből adódóan nagyon hosszú ideig kell adatokat gyűjteni (lehetőleg minél több targoncától, és gumi típustól), hiszen a kerekekből több száz munkaóra után kopik el csak egy milliméter. A gumik kopása azonban nem csak a munkaórák számától függ, hanem nagy hatással vannak rá a környezeti változók (talaj típusa, szennyeződések, időjárás), valamint a vezetés stílusa is. Ez alatt érthetjük például azt, hogy átlagosan mennyi súlyt cipel a targonca a villáján, vagy azt, hogy milyen gyakran kell fékeznie, illetve irányt váltania.

A valóságban tehát limitált mennyiségű historikus adat áll rendelkezésünkre a gumik elkopásáról, ezért szükség van arra, hogy a nagy felbontásban rendelkezésre álló adatokat is fel tudjuk használni a becsléshez. A kapott adatok a cég RX 20-as elektromos

targoncáitól tartalmaz adatokat (RX 20-16 és RX 20-20 altípusok), amelynek adatlapja letölthető [15]. A munkám végleges célja tehát egy RUL-modell felállítása, amely modell képes becslést adni arra, hogy az egyes targoncák kopó alkatrészeinek (pl. kerekeinek) az élettartamából még hány munkaóra van hátra. Ennek eredményeit a projekt következő fázisában egy felhő alapú feldolgozó egység hasznosítja majd, amely a betanított modell és a beérkező új adatok segítségével képes a kerék RUL értékek frissítésére bizonyos időközönként. Először azonban a munka első fázisában a már rendelkezésre álló adatok, adatbányászati eljárások és gépi tanulási módszerek segítségével kell egy modellt felállítani.

A továbbiakba a STILL által adott különböző adathalmazokat fogom bemutatni, majd az egyik adathalmazon végzett klaszterezési eljárás lépéseit, amelynek célja vezetési profilok megállapítása volt. Ezt követi a szerviz jelentéseken végzett szöveg analízis, amelyből az elkopás miatt lecserélt gumik dátumait nyertem ki. Utolsó lépésben a két adathalmazt kombinálva döntési fa alapú regressziós modelleket állítottam fel.

3.2 A meglévő adathalmazok áttekintése

Az úgynevezett *fleetmanager* adathalmazt a STILL targoncák vezérlő egységei (Main Controller Unit – MCU) gyűjtik a CAN (Controller Area Network) busz adatok folyamatos aggregálásával. Ez a vezérlő egység extraként kérhető a targoncákba a cégek által, így nem minden targoncánál áll rendelkezésre ilyen adat. Természetesen a korábban említett GSM modem ezeket az adatokat is visszajuttatja a STILL-hez. Egy *fleetmanager* bejegyzés alap esetben 10 percnyi CAN busz adat összegzésével készül el, de rövidebbek is lehetnek, ha a targonca vezetője egy 10 perces intervallum közepén állítja le a targoncát (pontosabban kiveszi az RFID kártyáját a targoncából). A bejegyzések tartalmazzák a mérési intervallum hosszát, valamint olyan alapvető adatokat, mint például, hogy mekkora távolságot tett meg a targonca, mi volt a maximális sebessége, mennyi energiát fogyasztott el és természetesen egy időbélyeget is. **Ezen az adathalmazon végeztem a klaszterezést, amelynek célja az egyes bejegyzésekhez vezetési profilok hozzárendelése, amely paraméterként használható a regressziós modellben.**

Szintén **fontos adathalmaz a szervizelési jelentések német nyelvű gyűjteménye**. Ez az erősen „zajos” adathalmaz több, mint 20 ezer targonca szerviz jelentéseit tartalmazza, amely sok belső használatra szánt változók (ezek a feladat szempontjából lényegtelenek is, csak a STILL számára bírnak jelentés tartalommal, pl.

alkatrész sorszámok) mellett rendelkezik 2 olyan oszloppal is, amelyben a szervízelést végző technikusok szövegesen leírják az elvégzett munkájukat (milyen problémát állapítottak meg, milyen javítást végeztek el). **A feladat ezen 2 oszlop segítségével azoknak a jelentések beazonosítása, ahol kerekek elkopása miatt kerékcsere történt azon targoncák egyikén, amelyhez rendelkezünk fleetmanager adatokkal.** A szervíz jelentésekhez természetesen tartozik dátum is, így ideális esetben előállítható egy olyan változó, amely számon tartja, hogy a targoncák kerekei hány munkanap után koptak el. Elegendően sok ilyen érték kinyerésével a jelentésekből felállítható egy olyan regressziós modell, ahol a fleetmanager adatok időbeli összegzett értékei a bemeneti változók, a becsült változó pedig a kerekek élettartama napokban kifejezve.

Ezen kívül **rendelkezünk még nyers CAN busz adatokkal is**, amelyet a STILL saját teszt pályáin gyűjtött. Mivel a CAN busz adatok időbeli felbontása sokkal nagyobb a fleetmanager adatokhoz viszonyítva, ez az adathalmaz pár nap hosszúságú vezetésből született. Ezen adatok rögzítésének célja az egyes vezetési profilok közötti különbségének felderítése volt. Tehát az adatok rögzítése során szándékosan eltérő szituációkba helyezték a STILL-nél a targoncát (hosszú egyenes vezetések, szűk raktári környezetben navigálás, teherautóból kipakolás során rámpán gyakori fel/lehajtás stb.). Ezt az adathalmazt egyelőre még nem használtuk fel a modellekhez, feldolgozása és kiértékelése folyamatban van.

Végül pedig rendelkezünk olyan adathalmazzal is, ami több, mint 700 targonca hibakódjait rögzíti. Ez egy igen nagy méretű adathalmaz, ugyanis teljesen normális, hogy a targoncák különböző hibakódokat rögzítenek működésük során. Ezen az adathalmazon nem én dolgozom, de a célunk vele egy RCA-t (hibaok analízist) végző modul felállítása, amely a meghibásodott targonca hibakódjaiból mintázatfelismeréssel képes megmondani a hiba okát, esetleg a hiba megjelenését előre jelezni is. A feladatot nehezíti a hibakódok nagy mennyisége működés közben, így egy komoly problémához tartozó hibakódok között számos másik hibakód is megtalálható.

3.3 Fleetmanager adatok csoportosítása vezetési profilokba

3.3.1 Vezetési profilok megkülönböztetése

A STILL szakértői szerint a CAN busz által mért paraméterek, mint például a nyomaték és fordulatszám, valamint a gumik kopása között nem lehet direkt kapcsolatot

felállítani. A CAN busz paraméterek értékei csak az idő egy adott pillanatára vonatkoznak, így hatásuk túl pici a kerékre. A szakértők ezért beazonosítottak néhány merőben eltérő használati esetet, amelyek eltérő szinten terhelhetik a targonca kerekeit. Ezek az alábbiak:

- Tolató vezetés: a targonca szűk helyen manőverezik, alacsony átlagsebességen.
- Rámpán vezetés: a targonca fel- és lemegy egy rámpán (például egy teherautó kirakodása során), így a súlyeloszlás gyakran változik.
- Gyors vezetés: ezt a profilt kevés irányváltás, nagyobb átlagsebesség és fordulatszám, alacsony nyomaték jellemezi.

Az egyes vezetési profilok hatását befolyásolhatja, hogy a targonca mekkora súlyt cipel a villáján, valamint a környezeti változók (talaj minősége és típusa, nedves-e a talaj, milyen szennyeződések találhatóak rajta stb.) is. A fleetmanager adatok klaszterekbe csoportosításának feladata ezen vezetési profilok igazolása, valamint extra magyarázó változók biztosítása a regressziós modellhez. A klaszterekbe csoportosítás annyiban tér el egy osztályozási feladattól, hogy nem rendelkezünk felcímkézett tanító adatokkal, így az algoritmusok pontosságát sem lehet kiértékelni mérőszámokkal. Ezeket a feladatokat hívják felügyelet nélküli gép tanulásnak (unsupervised machine learning) [16].

3.3.2 Az adathalmaz jellemzői

A fleetmanager adathalmaz, amelyen a végleges osztályozást végeztem 69 elektromos targonca bejegyzéseit tartalmazta. Az időbélyegek a rekordokon 2013.09.17. 12 órától 2016.08.28. 23 óráig tartanak, amely alatt 2 554 846 sornyi bejegyzés került rögzítésre. Az adathalmazban a következő 12 változó értékei találhatók meg:

- ***identifier***: A targoncát azonosító karaktersorozat.
- ***client_name***: A targoncát bérlő cég neve.
- ***timestamp***: A mérés időbélyege.
- ***readout_duration***: A mérés időtartama milliszekundum mértékegységben. Az összes időmennyiséget kifejező változó mértékegysége milliszekundum volt az adathalmazban. A további változók értékeit ezen változó és az időbélyeg által meghatározott időintervallumra kell érteni.

- ***drive_time***: Az az időtartam, amelyet mozgással töltött a targonca.
- ***lift_time***: Az az időtartam, amely alatt a targonca villáját súly terhelte.
- ***lift_drive_time***: Az az időtartam, amely alatt a targonca vagy mozgásban volt, vagy villáját súly terhelte. Ebből kifolyólag, ha egy targonca terhelés alatt mozgásban volt, akkor ennek a változónak az értéke kevesebb lesz, mint az előző kettő változó összege.
- ***distance***: A megtett távolság méterben kifejezve.
- ***max_speed***: A maximális elért sebesség, km/h mértékegységben kifejezve.
- ***number_of_direction_changes***: A menetirány váltások száma.
- ***consumed_amount***: Az elfogyasztott elektromos energia mennyisége.
- ***energy_unit***: Faktoriális változó, amely az előző változó mértékegységét határozza meg. Két lehetséges értéke az amperóra [Ah] és a kilowattóra [kWh].

3.3.3 Klaszterek optimális száma

Első lépésként kizárólag az adat alapján szerettem volna meghatározni a klaszterek optimális számát, mivel tudtam előre a bejegyzések nagy száma miatt, hogy a klaszterezéshez k-means algoritmust [17] szeretnék használni. A k-means működése pedig olyan, hogy előre meg kell adni hány klaszterbe szeretnénk csoportosítani az adatokat. Bár nincsen széles körben elfogadott módszer annak megállapítására, hogy egy adathalmaz hány klasztert tartalmaz, sok különböző metrika (index) létezik [18] amely szerint össze lehet hasonlítani két különböző klaszter számmal rendelkező csoportosítást. Léteznek belső indexek, mint például a klaszteren belüli átlagos távolság két pont között, és vannak külső indexek, amelyek a klaszterek közötti hasonlóságot mérik.

A különböző indexek kiszámolására egy specifikus R csomagot használtam (minden adatbányászati feladatot az ilyen feladatokra kifejlesztett R nyelven végeztem), amelyet direkt az optimális klaszter szám meghatározására készítettek. A tesztek futtatása előtt az adathalmazt leszűrtem csak a teljes, 10 perces bejegyzésekre, valamint kiválasztottam azokat a változókat, amelyek a vezetési profil szerinti csoportosításnál relevánsak lehetnek. Végül minden oszlop értékeit normalizáltam, hogy minden változó egyforma súllyal számítson klaszterbe soroláskor. A függvény számára a következő

paramétereket lehet megadni: magát az elő-feldolgozott adathalmazt, távolság számítási módszer két pont között (pl. euklideszi), módszer két klaszter távolságának meghatározására (pl. a távolság a két legközelebbi pont között), a kritérium, amellyel kiértékeljük a csoportosításokat (30 különböző index adható meg, plusz az „all” opció, amely kiszámolja mind a 30 indexet, és többségi szabály szerint hoz döntést végül), valamint a minimális és maximális klaszter számot, amelyre le kell futnia a csoportosításnak. Tehát ha minimum klaszter számosságnak 2 van megadva, maximumnak pedig 8, akkor 7 különböző csoportosítást fog elvégezni a kód, és 30 index szerint lehet kiértékelni, hogy melyik az optimális klaszter számosság a 7 közül. Mivel a szűrés után is több, mint 2 millió fleetmanager bejegyzés maradt, a 30 index kiszámítása rendkívül sokáig tartana egy számítógépen. A számítások gyorsítása érdekében véletlenszerű mintavételezést alkalmaztam, és az adatok 0.1%-át (kb. 2500 bejegyzést) használtam a számításokhoz, összesen 10 alkalommal.

A kapott eredmények vegyesek voltak. Az indexek között minden alkalommal voltak olyanok, amelyek 5, 7 vagy 8 klasztert tartottak optimálisnak. Az indexek többsége azonban a 2 és 3 klasztert tartotta ideálisnak. Négy alkalommal „nyert” a 2 klaszter a többségi szabály szerint, és hat alkalommal a 3 klaszter. A 3-nál több klasztert javasló indexekre lehet az a magyarázat, hogy az indexek alkalmazhatósága adathalmaz és paraméter függők, így bizonyos indexek kevésbé jó eredményeket adnak. Az erős megosztottságra a 2 és 3 klaszter között pedig az a legvalószínűbb magyarázat, hogy a 3. klaszter mérete jelentősen kisebb a másik két klaszternél. Ez nem meglepő, hiszen logikusnak tűnik, hogy a targoncák átlagosan viszonylag kevés időt töltenek rámpákon fel- és lehajtással.

3.3.4 Centroid-alapú és hierarchikus csoportosítások

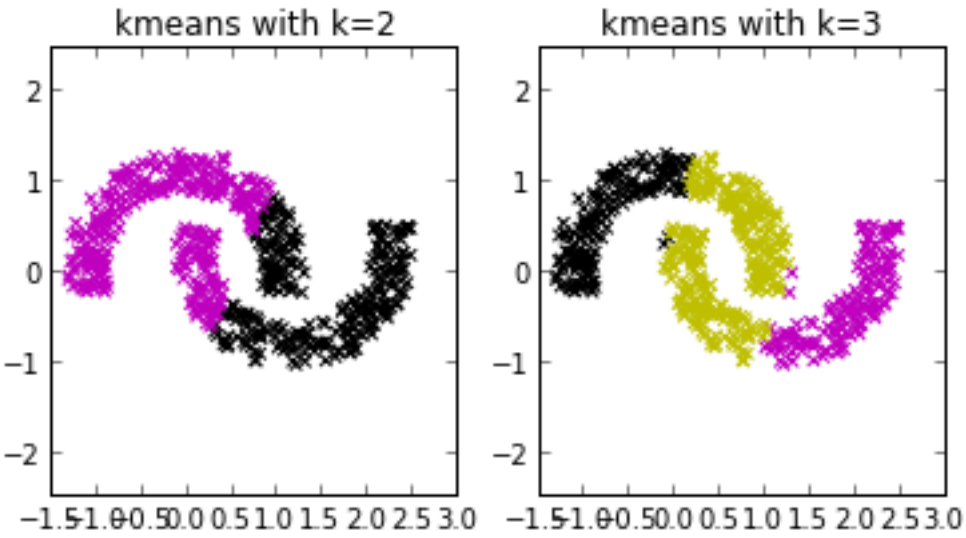
Az előző alfejezetben kapott eredményekkel nem voltam teljesen elégedett, mivel nincs egy konszenzus a kiértékelési indexek között. Ennek hatására kezdtem el utána járni, hogy milyen típusú klaszterezési algoritmusok léteznek még, és ezeknek mik az előnyei és hátránya.

A használni kívánt k-means csoportosítás egy általános rendeltetésű centroid-alapú klaszterezési eljárás. Az eljárást megvalósító implementációknak előnye, hogy nagy mennyiségű adat mellett is jól használhatók még, mivel a futási idejük az adatok számával egyenesen arányos (további paraméterek, amitől a futási idő függ még:

klaszterek száma, iterációs lépések száma, adatok paramétereinek száma). Sajnos számos hátránnyal is rendelkeznek, például azzal, hogy a csoportosítás végeredményét nagyban befolyásolni képesek a kezdeti véletlenszerűen kiválasztott klaszter középpontok. Ennek kiküszöbölésére léteznek tovább fejlesztett verziók [19], illetve a legtöbb adatbányászati *library-nél*, amely implementálja a k-meanst, bemeneti paraméterként meglehet adni, hogy az algoritmus hány véletlen kiindulási állást próbáljon ki. További hátránya a k-meansnek, hogy nem képes konkáv „alakú" klaszterek megtalálására, valamint törekszik arra, hogy a klaszterek kiegyenlítettek legyenek (hasonló elemszámú legyen mindegyik), amely sok használati esetben nem helytálló. A mi esetünkben például az előzetes elvárásunk az, hogy a gyors vezetési profilban az átlagos megtett távolság nagy, míg az irányváltások száma kicsi, míg a tolató vezetési profilban ennek a fordítottját várjuk. Egy harmadik gyenge pontja a k-meansnek (amely az előző kettő problémával ellentétben, az itteni használati esetben nem jelent gondot) az, hogy a változók számának növekedésével jelentősen romlik a hatékonysága.

A klaszterezési eljárások másik széles körben használt csoportját a hierarchikus klaszterezések alkotják. Ezek az eljárások leggyakoribb kiindulása állapotában minden adatpontot saját klaszterbe rendel, majd a további lépésekben a két legközelebb álló klasztert összevonja, amíg csak 1 klaszter marad, amelybe az összes bejegyzés beletartozik. Itt az egyes eljárásokat azt különbözteti meg, hogy milyen függvényt használnak a pontok közötti távolságok számítására, valamint milyen metodikát használnak a klaszterek közötti távolságok megállapítására (két legközelebbi pont, középpontok távolsága, két legtávolabbi pont stb.). Ellentétes megközelítésére is léteznek implementációk, ahol a kiindulási állapot egy klaszterből áll, és minden lépésben úgy kerülnek felosztásra a klaszterek, hogy a lehető legtávolabb legyenek egymástól az új klaszterek. A végállapotban minden bejegyzés saját klaszterrel rendelkezik. Hierarchikus klaszterezésnél a felhasználó feladata szakértelme segítségével a megfelelő felosztás mellett dönteni (mennyi a klaszterek végső száma), miután lefutott az algoritmus. Ezen algoritmusok gyengesége épp az, ami a k-means erőssége, vagyis a futási idejük rendkívül gyorsan növekszik az adatpontok számával (n^3 vagy 2^{n-1} nagyságrendjében). Ráadásul ezek az eljárások a mohó algoritmusra építenek, vagyis mindig a lokálisan optimális lépést hajtják végre, így nagy adathalmazok esetén nagyon kicsi arra az esély, hogy a globális optimum jön ki eredményül. Nem képesek jól kezelni a kilógó adatokat sem.

A hierarchikus eljárások viszont nem rendelkeznek a k-means gyengeségeivel, így a két fajta klaszterezési eljárás jól kiegészíti egymást. A k-meansre lehet úgy gondolni szemléletesen, hogy az adathalmazunkat papírból kivágott körökkel szeretnénk minél pontosabban lefedni, és az egy kör alá eső pontok tartoznak egy klaszterbe. A 3.1. ábrán jól látható, hogy konkáv adathalmazoknál ez nem vezet jó eredményre.



3.1. ábra: K-means konkáv adathalmazon [20]

A két fajta eljárás ötvözése viszont egy robosztus megoldást biztosíthat. Az adathalmaz „formája” megmarad, ha a k-means sok kis lokális klasztert hoz létre (kis körökkel fedjük le a pontokat), amely kimenetét pedig használhatjuk egy összevonó jellegű (bottom-up) hierarchikus algoritmus bemenetének. Ezzel mindkét eljárás legnagyobb gyengeségeit küszöböljük ki, hiszen a hierarchikus algoritmus számára drasztikusan lecsökkennek az adatpontok száma.

3.3.5 Adatok tisztítása, értékes változók előállítása

Klaszterezés előtt természetes, hogy érdemes az adathalmazt megtisztítani a hibás és/vagy kilógó bejegyzésektől, amelyek az eredményeket torzítanák. Emellett szintén célszerű az adathalmaz alapos megismerése (pl. vizualizáció segítségével), hogy a meglévő adatokból a leglényegesebb információkat kinyerhessük.

3.2. ábra: Fleetmanager adathalmaz előkészítése a klaszterezésre

Ennek a folyamatban a menete látható a 3.2. ábrán. A fleetmanager adathalmaz (amelynek változóit már bemutattam 3.3.2-ben) esetében a lényegi információ kinyerése azt jelentette, hogy kiszűrtem az összes olyan bejegyzést, ahol a mérés időtartama 1 percnél kevesebb volt. Továbbá, minden olyan sor is szűrésre került, ahol a vezetési idő és a mérés időtartamának aránya 5%-nál kevesebb volt. Így az adatok 8%-át hagytam el, de előzetes vizsgálataim alapján erre szükség volt, mert sok bejegyzés létezett, ahol a targonca tétlen állapotban volt a mérés alatt, amely így nem releváns a gumikopás szempontjából (valamint vezetési profilnak se nevezhető a tétlen állapot). A „spektrum" másik végét is megvizsgáltam, azaz minden használni kívánt változó értékeinek felső 5%-át nagy felbontás mellett megnéztem. Ahol nagy ugrást tapasztaltam a változó értékében, a határ feletti bejegyzéseket szintén elhagytam (pl. 653 bejegyzést, ahol az irányváltások száma 135-nél több volt). Ezzel az adatok további 1%-a került szűrésre.

Ha megnézzük a ***lift_drive_time*** változó jelentését, akkor láthatjuk, hogy az értéke erősen korrelált mind a ***drive_time*** és a ***lift_time*** változó értékével. Korrelált változók nincsenek nagy hatással a klaszterező eljárásokra, viszont regressziós modellek esetében nehezíti a modellek kiértékelését, könnyen vezethet overfittinghez[1], vagy akár rossz predikciókhoz is, ha a korreláltság nem áll fenn az új adatoknál. Ezért tartottam szükségesnek előállítani az alábbi 3 változót a meglévőkből:

[1] Az a jelenség, ahol a statisztikai modell túl jól illeszkedik a tanuló adatokra, mert „rátanul" a tanuló adatokban található zajra is. Általában feleslegesen túl komplex modelleknél fordul elő (túl sok paraméter, túl mély döntési fa, túl sokáig tanított neurális háló validáció nélkül stb.). Ezek a modellek új adatokon jelentősen rosszabbul teljesítenek, ezért aktívan védekezni szoktak ellene.

- ***drive_only_time***: Az az időtartam, amelyet mozgással töltött a targonca, miközben nem terhelte súly a villáját.
- ***lift_only_time***: Az az időtartam, amely alatt a targonca villáját súly terhelte, de nem volt mozgásban.
- ***lift_and_drive_time***: Az az időtartam, amely alatt a targonca mozgásban volt, miközben súly terhelte a villáját.

Mivel nem minden bejegyzés időtartama 10 perc, ezért szükség volt bizonyos változók normalizálására a ***readout_duration*** változó értékével. Így lett a megtett távolságból átlagsebesség (km/h mértékegységgel), vezetési időből a vezetési arány, emelési időből az emelési arány, valamint az irányváltoztatások száma is módosult az alábbi módon:

$$NormDirChange = NumberOfDirectionChanges * \frac{\max(ReadoutDuration)}{ReadoutDuration}$$

Két fontos feladat van még az előkészítés során. Először is a klaszterezéshez használni kívánt változókat új adatkeretbe (data frame) kell helyezni: ***average_speed, drive_only_ratio, lift_only_ratio, lift_and_drive_ratio, norm_number_of_dir_change*** és ***consumption_rate***. Ez az adatkerek lesz a k-means bemenete. Megfontoltam a ***max_speed*** változó használatát is, de végül úgy döntöttem, hogy értéke csak egy pillanatnyi időpontot jellemez, nem írja le jól a teljes mérés időtartamát. Továbbá, használata redundáns is lett volna az átlagsebesség alkalmazása mellett. A ***consumption_rate*** változó az elfogyasztott energia mennyiség idővel való leosztása után állt elő, Watt mértékegységre átváltva. Sajnos a fogyasztott energiamennyiség két mértékegysége (kWh és Ah) nem váltható át egymásba, mivel a targoncák akkumulátor feszültsége nem konstans a működés során. A bejegyzések körülbelül két-harmada (1.5 millió sor) kWh mértékegységben rendelkezett a ***consumed_amount*** változóval, ezért amellett döntöttem, hogy két különböző klaszterezést fogok csinálni. Az egyikben nem lesz benne a fogyasztás egyáltalán, csak az első 5 változó, míg a második klaszterezésben csak azokat a bejegyzéseket fogom használni, amelynél a fogyasztás Wattban kiszámítható.

Az előkészítés utolsó lépése a változók átskálázása, hogy mindegyik átlaga 0, szórása pedig 1 legyen. Erre szimplán azért van szükség, mert az egyes változók értékei különböző nagyságrendbe esnek. Ha például vesszük a fogyasztás átlagát (1950 Watt) és

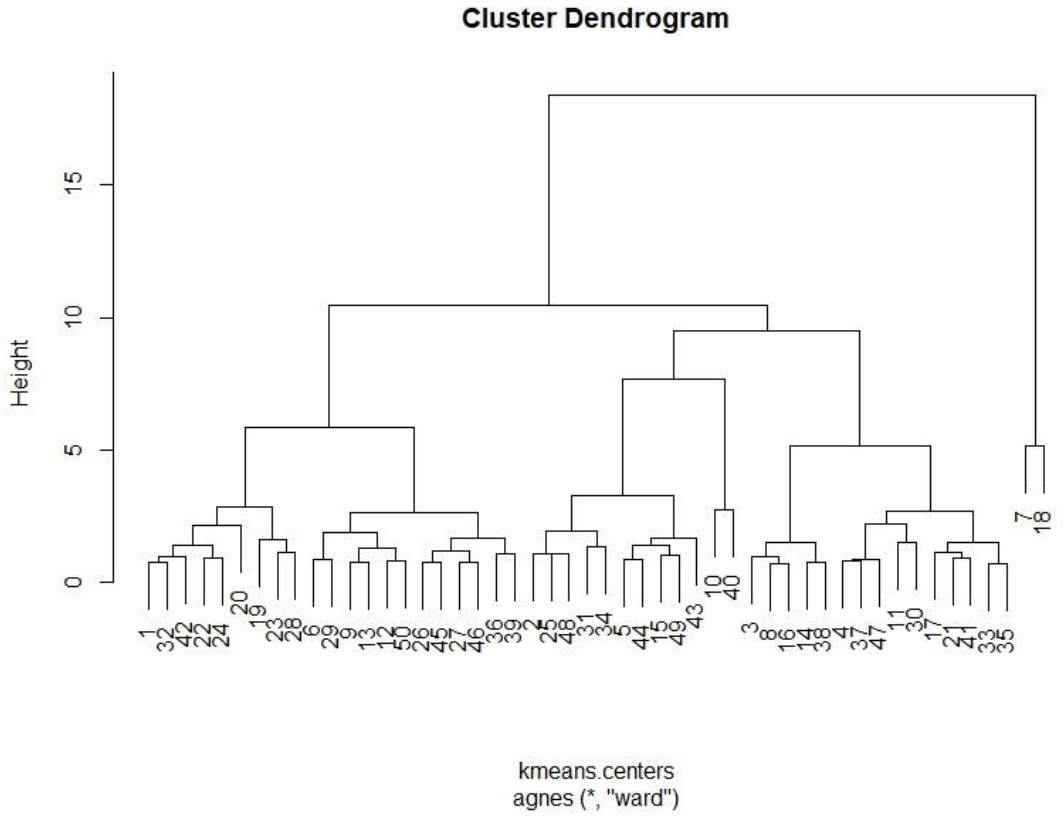
az átlagsebességet (3,31 km/h), akkor könnyen belátható, hogy skálázás nélkül a k-means sokkal nagyobb súllyal venné figyelembe a fogyasztás értékét (az euklideszi távolságok számítása miatt). Ezzel szemben az elvárt működés az, hogy mindegyik változó egyenlő mértékben számítson a klaszterek kialakításánál. Az átskálázás természetesen nem jár adatvesztéssel, hiszen a tört számokat tárolni tudjuk kellő pontossággal.

3.3.6 A csoportosítások eredményei

Először tehát a k-means csoportosítást futtattam az adatokon. Empirikus módszer alapján 50 klasztert adtam meg a k-means paraméterének, 20 véletlenszerű kiindulási állapot kipróbálásával. A maximum iterációs lépések száma 100 volt. Az algoritmus 20 különböző kiindulási állapotból lefuttatására azért van szükség, mert előfordulhatnak olyan kezdeti állapotok, amelyek hatására az algoritmus egy lokális minimumra konvergál csak, amely messze van az optimális felosztástól. Az implementált algoritmus a 20 eredmény közül azt választja legjobbnak, amelynél a klasztereken belüli varianciák összege a legkisebb. A k-means kimenete a legjobb csoportosításhoz tartozó 50 klaszter középpontja volt, amely (vagy a centerek közötti távolsági mátrix) a bemeneti paramétere volt a hierarchikus klaszterezésnek. Ennél a lépésnél a fő kiválasztandó paraméter a *linkage criteria* volt, amely azt határozza meg, hogy a klaszterek közötti távolság milyen metrika szerint legyenek számolva. K-means alkalmazásával a hierarchikus klaszterezés futási ideje drasztikusan lecsökkent (2.5 millió lépés helyett csak 50), ezért lehetőségem volt különböző kritériumok kipróbálására:

- **Minimum/single linkage**: két klaszter távolságának a minimum távolságot vesszük olyan két pont között, amelyek nem ugyanabban a klaszterben vannak.
- **Weighted/average linkage**: két klaszter távolságának az eltérő klaszterbe tartozó pontok közötti távolságok számtani átlagát vesszük.
- **Centroid linkage**: két klaszter távolságának a centerek euklideszi távolságát vesszük.
- **Ward linkage**: ennél a kritériumnál azok a klaszterek kerülnek összevonásra, amelyek összevonásával a klasztereken belüli varianciák összegének növekedése a minimális lesz.

Hierarchikus klaszterezések egyik jellemző végterméke a dendrogram. A dendrogram egy fagráf, amely megmutatja, hogy az egyes iterációk során mely klaszterek kerültek összevonásra. A gráfot tetszőleges szinten elvághatjuk, hogy megkapjuk a kívánt klaszterszámú csoportosításunkat. A fagráfok megvizsgálásával látható volt számomra, hogy a *linkage criteria* erősen befolyásolja a csoportosítás végeredményét, így végül azt a módszert választottam, amely az előzetes elvárásokhoz legjobban illő felosztást eredményezte. A Ward-féle kritérium [21] eredményét használtam így a továbbiakban, amely 3 klasztere közül 2 elemszáma hasonló mértékű a fagráf alapján (gyors és tolató vezetési profil), míg a harmadik számossága jelentősen kisebb (rámpa vezetés). A 3.3. ábrán látható a gráf, amely alján a k-means klaszterek sorszáma látható. A koordináta tengelyen az összevonáshoz használt kritérium értéke látható minden egyes szinten. Minél nagyobb a magasság, annál kevésbé hasonlítanak egymásra az összevont klaszterek.



3.3. ábra: A Ward-féle hierarchikus klaszterezés dendogramja („fogyasztás” változó nélküli eset)

A dendrogramot 3 klaszternél elvágva megkapjuk a csoportosítást. A k-means 50 klaszteréből 42% tartozik az első klaszterbe, 54% tartozik a másodikba, és 4% a harmadik klaszterbe.

A hierarchikus klaszterezés eredményét visszavetítve az eredeti fleetmanager adatokra azt kapjuk, hogy 47,15% tartozik az első klaszterbe, 51,85% a másodikba, és végül 1% tartozik a harmadik vezetési profilba a „fogyasztás” változó nélküli klaszterezés eredményeképp.

A fogyasztást figyelembe vevő csoportosításnál is hasonló eloszlású klasztereket kaptam (sorrendben 43,36%, 54,21%, 2,42%). Az átskálázás nélküli eredeti adathalmazon a klaszterek hozzárendelése után kiszámoltam a klaszterek középpontjait is. Ez a csoportosítás legfontosabb végeredménye, hiszen a tervek szerint a regressziós modell új adatok esetén a klaszter középpontok alapján dönti el, hogy az új bejegyzés melyik vezetési profilba tartozik (melyikhez a legkisebb az euklideszi távolsága). A középpontok láthatók a 3.4. ábrán.

	average_speed	drive_only_ratio	lift_only_ratio	lift_and_drive_ratio	norm_number_of_dir_change
cluster1_center	4.236390	0.627117127	0.01284471	0.1538396	30.51334
cluster2_center	2.515846	0.351867247	0.02248927	0.2899064	45.42963
cluster3_center	1.915398	0.006019873	0.54187696	0.4492873	27.84089

	average_speed	drive_only_ratio	lift_only_ratio	lift_and_drive_ratio	norm_number_of_dir_change	consumption_rate
kwh.cluster1_center	3.370215	0.5061513	0.02422862	0.2119591	39.20716	1962.308
kwh.cluster2_center	3.319263	0.4599069	0.01999138	0.2403567	37.66704	1973.056
kwh.cluster3_center	2.288088	0.4763199	0.12524455	0.1754710	25.41372	1352.129

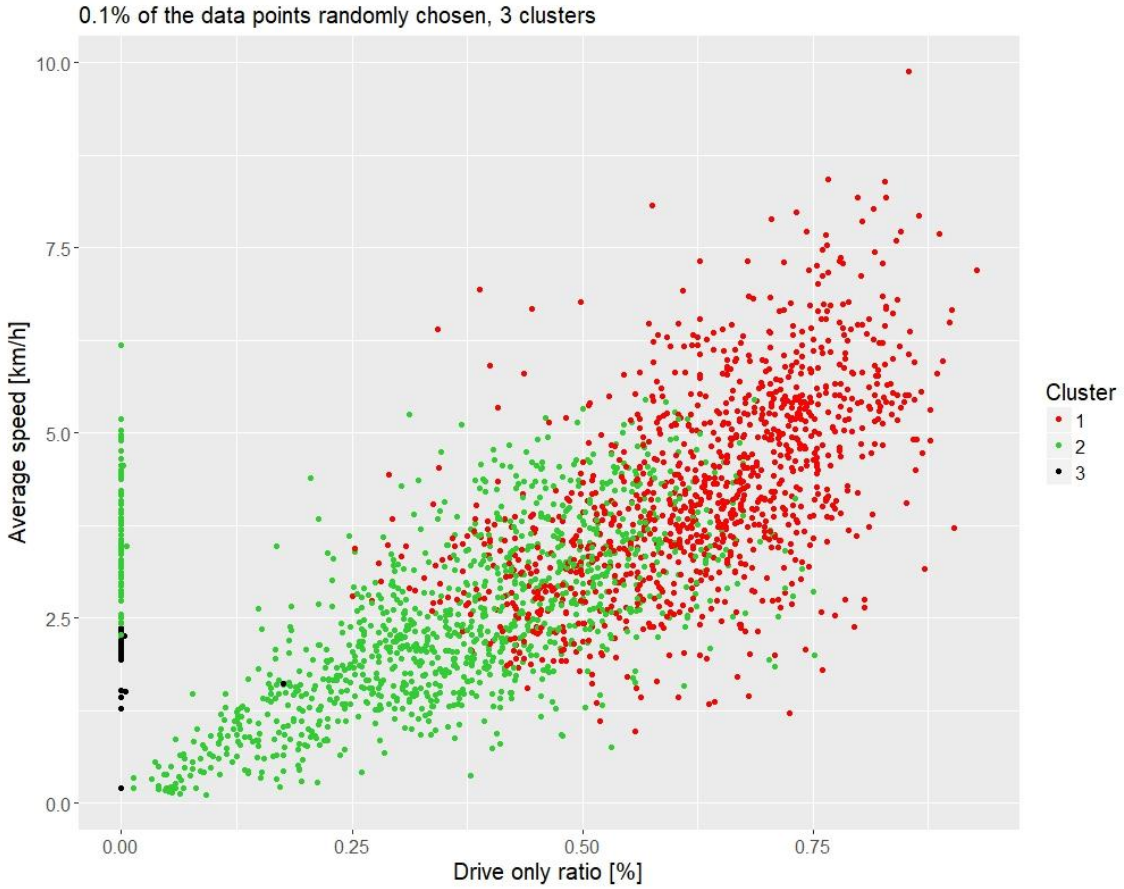
3.4. ábra: A klaszterek középpontjai

Jól látható, hogy a fogyasztás nélküli esetben sikerült a k-means gyengeségét kiküszöbölni, és nem olyan centereket kaptunk, amelyek lineárisan összefüggenek. Az első klaszternek a legmagasabb az átlagsebessége és vezetési aránya, így ehhez rendeljük a gyors vezetési profil címkéjét (adatok ~47%-a). Mivel tudjuk, hogy a harmadik profil számossága csak 1% körül van, ezért mindenképp ezt tekintjük a rámpás vezetési profilnak, amelyet a vezetési és emelési arányok is igazolni látszanak. Az utolsó klaszter fog tartozni így tolató vezetési profilhoz, amelynek értékei többnyire az előző két klaszter közé esnek, kivéve az irányváltások száma, amely ezen klaszterben a leggyakoribb (adatok ~52%-a).

Bár a dendrogramok jellege nem változott a fogyasztást figyelembe vevő csoportosításoknál, az eredmény itt sokkal kevésbé alakult az elvártak szerint. A 3.4. ábra alsó része szintén a Ward-féle kritériummal kapott eredményt mutatja, ahol a klaszterek eloszlása rendre 43.5%, 54% és 2.5% volt az eredeti fleetmanager adathalmazban. Jól látható, hogy az első két klaszter középpontjai között nincsen jelentős eltérés. Az okát ennek sajnos nem tudjuk teljes bizonyossággal megállapítani. Az 5 változós csoportosítás

2,3 millió bejegyzés alapján készült, míg a fogyasztást is figyelembe vevő kb. 1,5 millió adatpontból. Nyilvánvaló, hogy a változók száma és a bejegyzések száma közötti arány növekedésével a k-means használhatósága romlik, de ebben az esetben ez még nem volt számottevő véleményem szerint. Próbáltam számos különböző paraméter kombinációval is futtatni az algoritmust, de nem tapasztaltam jelentős javulást egyik esetben sem. Ennek egy lehetséges okát fogom részletezni a 3.7. ábra elemzésénél.

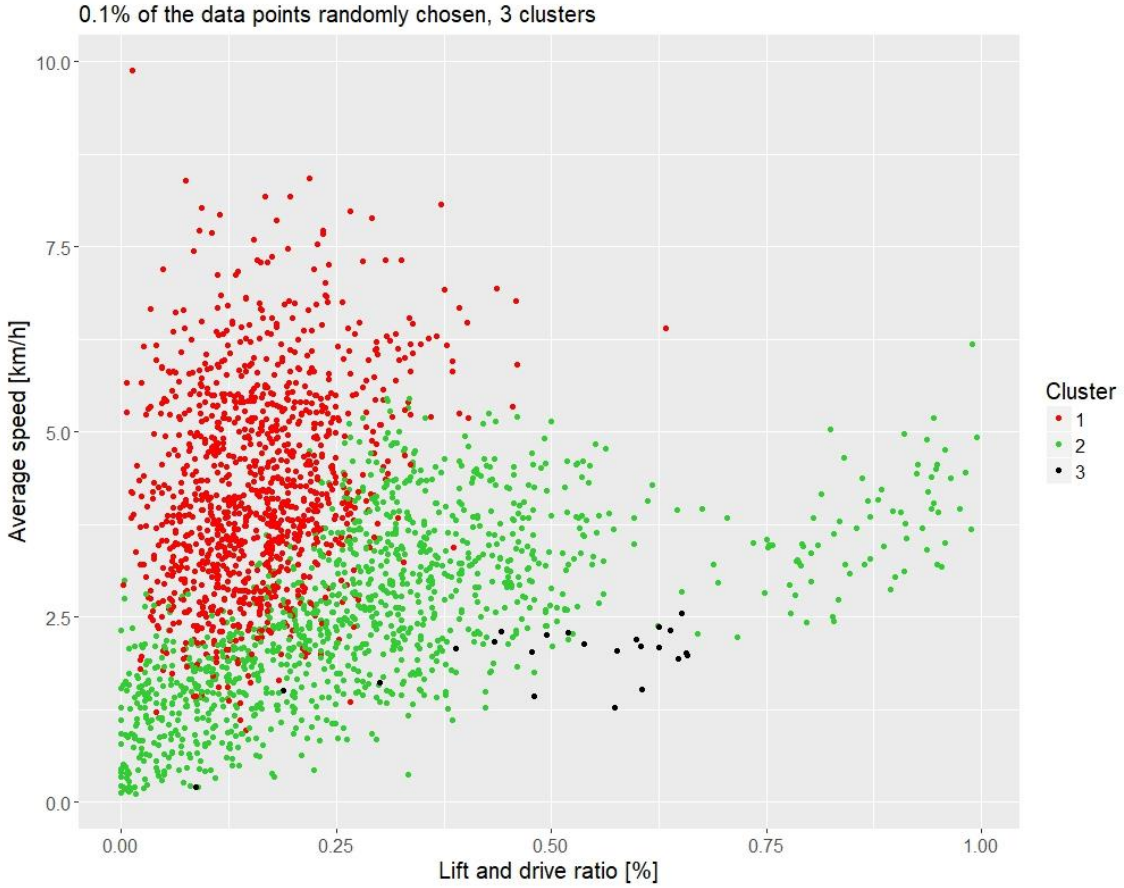
A klaszterezés sikerességét vizsgálva számos változó kombinációval készítettem szórásdiagramokat, valamint sűrűségfüggvény-diagramokat. Szórásdiagramoknál véletlenszerűen vettem az adatok 0,1%-át (2348 pontot) 5 változós esetben, és 0,2%-át (2948 pontot) 6 változós esetben. Erre természetesen azért volt szükség, hogy átlátható diagramokat kapjak.



3.5. ábra: Átlagsebesség változása vezetési arány mellett (5 változós eset)

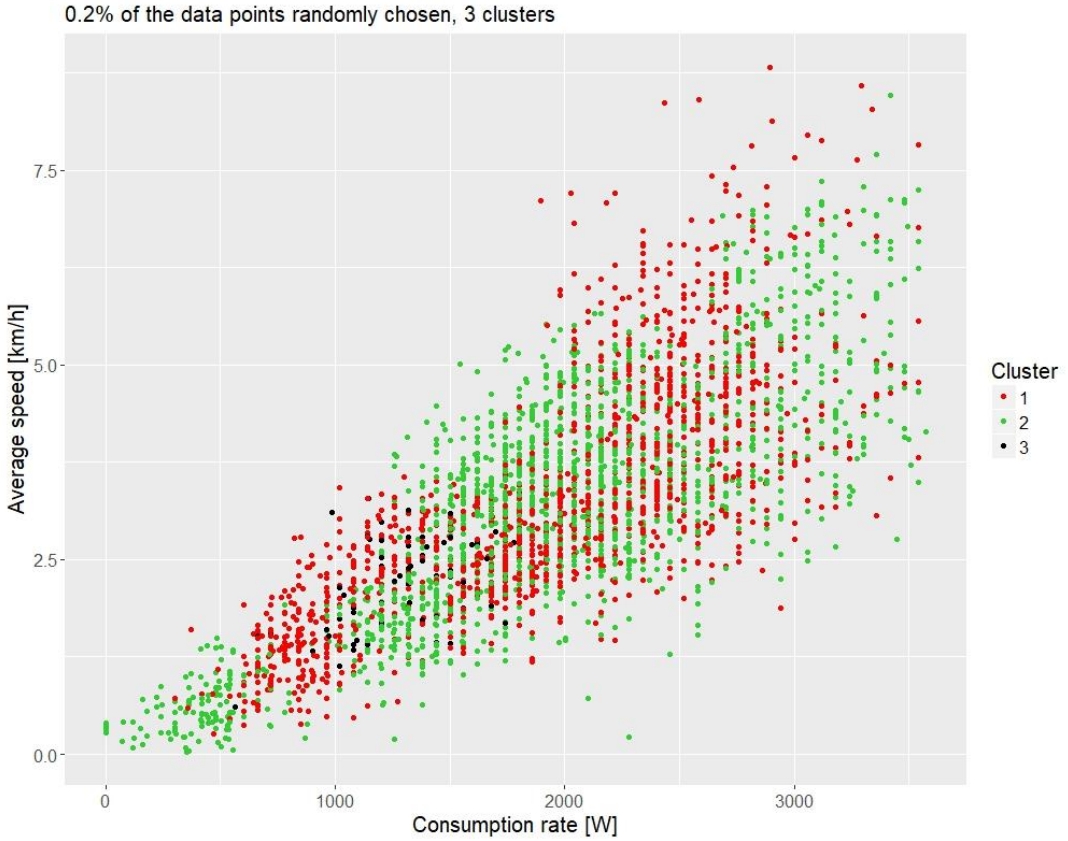
A 3.5. ábrán jól látható a vezetési arány és az átlagsebesség közötti lineáris kapcsolat. Ezen kívül az adatpontok klaszterei relatíve jól elkülönülnek egymástól úgy is, hogy az 5-ből csak 2 változót ábrázol a diagram. A kép bal szélén látható pontok első ránézésre hibásnak tűnhetnek (0 vezetési arány mellett 0-nál nagyobb átlagsebesség), de

annál a bejegyzéseknél garantáltan cipeltek súlyt is a targoncák vezetés közben. Mivel ugyanazokon az adatpontokkal készültek, ezt jól alátámasztja a 3.6. ábra is, ahol az átlagsebesség a vezetési és emelési arány mellett lett ábrázolva. A 3.7. ábra jól mutatja, hogy a fogyasztás változóval készített csoportosítás kevésbé lett sikeres. A klaszter középpontokkal összhangban azt tapasztaltam, hogy ennél az esetnél a klaszterek sokkal kevésbé különülnek el, mind a szórásdiagramokon, mind a sűrűségfüggvények esetében.



3.6. ábra: Átlagsebesség változása a vezetési és emelési arány mellett (5 változós eset)

Külön érdekesség, hogy a „fogyasztás” változó volt az egyetlen, ahol megfigyelhető az értékek „kvantáltsága”. Ez a fogyasztás sűrűségfüggvényén is jól látszik (recés alak). Ennek oka az eredeti ***consumed_amount*** változóban rejlik, amely egyszerre tárolt Ah és kWh mértékegységű értékeket 2 tizedesjegy pontossággal, miközben a kWh mértékegységű bejegyzések értéktartománya 0-tól 0,59-ig terjedt. Ez a nem szerencsés mértékegység/pontosság választás az oka annak, hogy rengeteg 10 perces bejegyzés rendelkezik pontosan ugyanazzal a ***consumed_amount*** értékkel. Sajnos a változó felbontása nem elég nagy. A „fogyasztás” változó ennek eredményeképp kapott egy 60 W-os kvantáltságot, amely visszaköszön a szórásdiagramon (3.7. ábra) is.

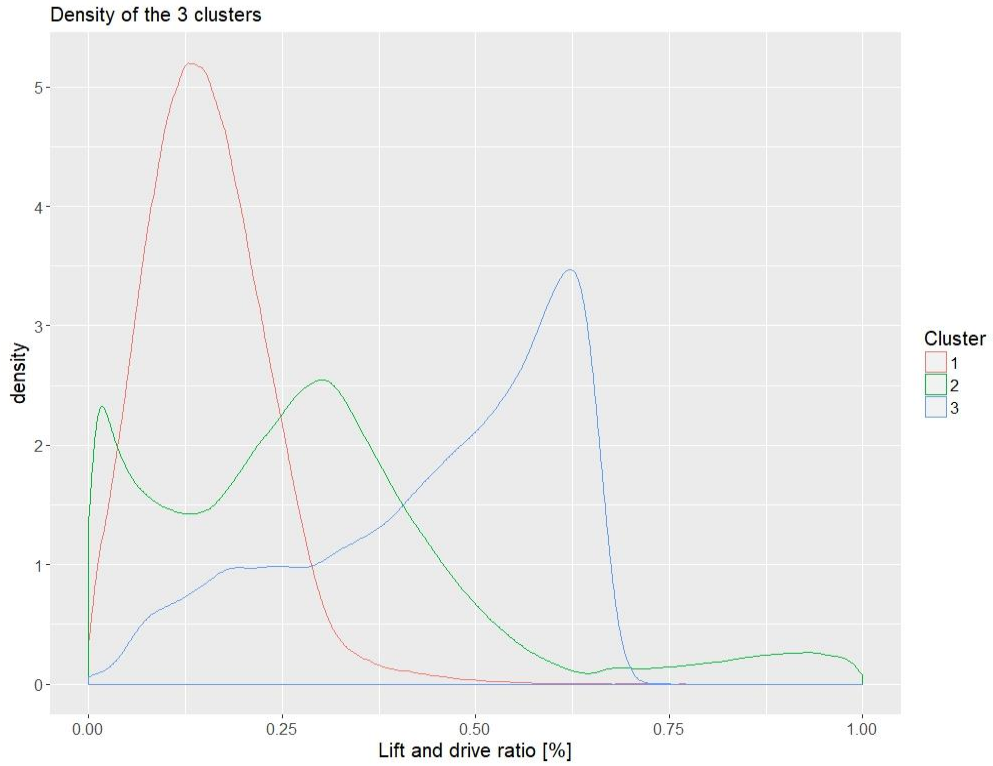


3.7. ábra: Az átlagsebesség és a fogyasztás lineáris kapcsolata (6 változós eset)

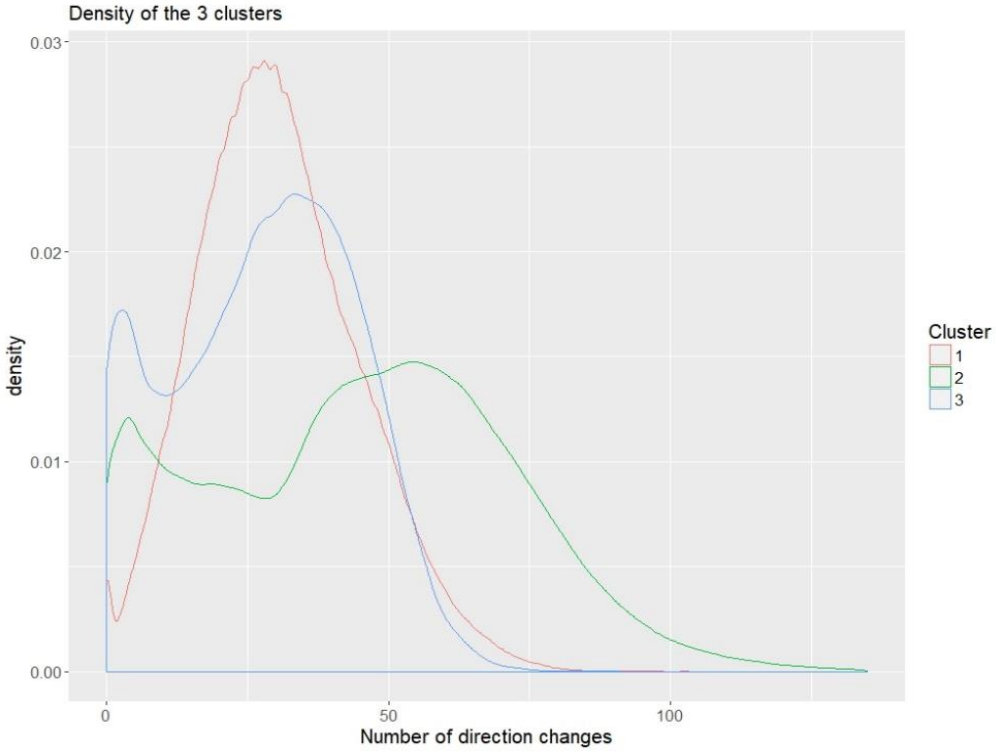
Ha egy fogyasztás értéke nem 60-nal osztható szám lett, akkor az azt jelenti, hogy a mérés időtartama 10 percnél rövidebb volt. Ezért látható a szórásdiagram bal oldalán olyan pontok, amely nem esnek egy függőleges vonalra a többi bejegyzéssel, míg a jobb oldalon már nincs ilyen pont. Véleményem szerint a fogyasztási változó kvantáltsága lehetett a fő ok, ami miatt a k-means algoritmus nem tudott rendesen konvergálni egy optimális megoldásra.

Az átlagsebesség, vezetési arány és emelési arány sűrűségfüggvényei normális eloszláshoz közelítő alakkal rendelkeznek mind a 3 klaszter esetében. Az emelési és vezetési arány esetében ez csak az első klaszter esetében igaz, de a csúcsértékek mégis szépen elkülönülnek, ahogy azt a 3.8. ábra mutatja. A jól elkülönülő csúcsértékek az adathalmaz csoportosítható tendenciájára enged következtetni. Kicsit eltérő a helyzet az irányváltozásoknál (3.9. ábra), ahol az első klaszter eloszlásának alakja is torzul egy picit, valamint a másik két klaszter eloszlása egész nagy mértékben hasonlít egymásra. A harmadik (kis méretű) klaszter azonban így is rendelkezik létjogosultsággal, hiszen ahogy az a középpontokból is leolvasható, az emelési aránya és vezetési aránya rendkívül eltérő a másik két klaszterhez képest. Az eredmények kiértékelése után azt állapítottam meg,

hogy a továbbiakban az 5 változós csoportosítás eredményeit érdemes felhasználnom, amelynek a skálázott (0 átlagú, 1 szórású) klaszter középpontjait így elmentettem egy *csv* fájlba.



3.8. ábra: Emelési és vezetési arány sűrűségfüggvénye (5 változós eset)



3.9. ábra: Irányváltoztatások sűrűségfüggvénye (5 változós eset)

3.4 Szerviz jelentések szöveges analízise

A szerviz jelentések adathalmazának célja az volt, hogy a regressziós modellhez elő tudjam állítani az eredmény változót. Ez a változó nem más, mint a kerekek élettartama valamilyen nagy felbontású idő mértékegységben (órák, napok). Az adathalmaz 311 031 szerviz jelentést tartalmaz, de ez sajnos gyorsan lecsökkent 2 536-ra, miután azonosító alapján szűrtem arra a 69 targoncára, amelyhez rendelkezünk fleetmanager adatokkal is. Az adathalmaz 41 oszloppal rendelkezik, de mint ahogy említettem már, ezek nagy része sajnos számunkra nem releváns információt tartalmaz, vagy többnyire nincs érték hozzárendelve. A következő 8 változót választottam ki további felhasználásra: targonca azonosítója, gyártási év, szerviz típus (3 számjegyű integer), város (ahol a targoncát használják), bérlő cég, szerviz komment (a szöveg analízis tárgya), valamint a szerviz kezdeti és vég dátuma. További 41 bejegyzés került ezután szűrésre, mivel a szerviz komment változójuk üres volt.

A szöveges mező feldolgozása előtt megvizsgáltam a szerviz típusnak gondolt oszlop értékeit (eredeti német neve *BewArt*), de sajnos nem tapasztaltam semmilyen összefüggést az oszlop értéke és a szerviz során elvégzett javítás típusa között. Szükségesnek gondoltam ezen eshetőség kizárását, mert ha a kerék cseréket beazonosította volna egy bizonyos 3 számjegyű kód, akkor nem lett volna szükség a szöveges analízisre. A továbbiakban ismertetem a szöveg feldolgozásának általános lépéseit, amely folyamatábrája a 3.10. ábrán látható.

3.10. ábra: A szöveg feldolgozásának folyamata

3.4.1 Szöveg feldolgozásának folyamata

A szöveg analízis első nagy lépése az úgynevezett *document feature matrix* (DFM) előállítása. Ennek a mátrixnak annyi sora van, ahány bejegyzésünk (szerviz kommentünk) van, és annyi oszlopa ahány egyedi szó megtalálható a teljes korpuszban (vagyis a szerviz kommentek összességében). A mátrix N. sorának és M. oszlopában lévő szám adja meg azt, hogy az N. dokumentumban (szerviz kommentben) hányszor szerepel a korpusz M. szava. Ez az adatstruktúra jó bemenetként szolgál gépi tanulási algoritmusok számára, valamint nagy méretű korpusz esetén a további kézi feldolgozáshoz is elengedhetetlen. Egy kis része látható az általam előállított DFM-nek a 3.11. ábrán. A DFM elkészítéséhez a quanteda R csomag [22] funkcióit használtam.

	fahrzeugubergab	ladegerat	betrieb	genomm	scheinwerf	eingestellt	ruckfahrsensor
text1	1	1	1	1	1	1	1
text2	0	0	0	0	0	0	0
text3	0	0	0	0	0	0	0
text4	0	0	0	0	0	1	0
text5	0	0	0	0	0	0	0
text6	0	0	0	0	0	0	0
text7	0	0	0	0	0	0	0

3.11. ábra: DFM első néhány sora és oszlopa

A DFM előállításának első lépése a szerviz kommentek tokenizálása, vagyis a szöveg komponensekre (ez esetben szavakra) való felbontása. Ezzel a lépéssel elhagyható az információt nem tartalmazó *whitespace* karakterek, számjegyek, írásjelek és központozások. Előfordultak olyan esetek, ahol nem sikerült minden számjegyet vagy írásjelet eltávolítani a komponensekre bontással, mert ezek a karakterek szavak részei voltak szóközök nélkül. Ezeknél az eseteknél reguláris kifejezéseket alkalmaztam, amelyek extra szóközöket szúrtak be, így a tokenizálás újra futtatásával már szűrésre kerültek. A komponensekre bontás végterméke látható a 3.12. ábrán.

```
Component 2492 :
[1] "gewaltschad"   "schutzblech"   "hydraulikleit" "gabel"         "neu"           "anschweis"

Component 2493 :
[1] "reif"

Component 2494 :
[1] "notausschalt" "defekt"       "armlehn"      "zerlegt"      "schalt"       "erneuert"

Component 2495 :
[1] "durchfuhr"     "turnusmass"    "wartungsarbeit" "jahrlich"      "fem"           "prufung"
```

3.12. ábra: A tokenizálás eredménye dokumentumokra bontva

Ebből már elő állítható lenne egy DFM, de még sok redundáns token, valamint nem sok információt tartalmazó kötőszó is szerepelne benne, amelyeket el lehet belőle távolítani. Ezért az alábbi lépéseket végeztem még el a DFM előállítása előtt:

- Mindegyik tokent kisbetűs formára alakítani (tokenek alapból kis-nagybetű érzékenyek).
- A német nyelvre jellemző gyakori kötőszavak eltávolítása a tokenek közül. A gyakoriságukból kifolyólag nagyon kevés információt hordoznak. A quanteda csomag beépítetten tartalmazza a gyakori kötőszavakat számos nyelvre. Példaként láthatók a 3.13. ábrán azok a magyar szavak, amelyeket a quanteda képes kiszűrni. Egyedi listát is megadhatunk, ha a csomagban tárolt kötőszavak valamelyikéről úgy gondoljuk, hogy az adott esetben relevanciával bír.
- Szintén támogatott funkció a szavak szótőre való visszabontása. Egy szó különböző ragozott alakjai helyett így csak a legegyszerűbb alakjából lesz token, amellyel tovább tudjuk csökkenteni a tokenek számát információ vesztés nélkül.

```
> stopwords(kind = "hungarian")
  [1] "a"          "ahogy"      "ahol"       "aki"        "akik"       "akkor"      "alatt"      "által"      "általában"  "amely"      "amelyek"
 [12] "amelyekben" "amelyeket"  "amelyet"    "amelynek"   "ami"        "amit"       "amolyan"    "amíg"       "amikor"     "át"         "abban"
 [23] "ahhoz"      "annak"      "arra"       "arról"      "az"         "azok"       "azon"       "azt"        "azzal"      "azért"      "aztán"
 [34] "azután"     "azonban"    "bár"        "be"         "belül"      "benne"      "cikk"       "cikkek"     "cikkeket"   "csak"       "de"
 [45] "e"          "eddig"      "egész"      "egy"        "egyes"      "egyetlen"   "egyéb"      "egyik"      "egyre"      "ekkor"      "el"
 [56] "elég"       "ellen"      "első"       "először"    "előtt"      "első"       "én"         "éppen"      "ebben"      "ehhez"      "emilyen"
 [67] "ennek"      "erre"       "ez"         "ezt"        "ezek"       "ezen"       "ezzel"      "ezért"      "és"         "fel"        "felé"
 [78] "hanem"      "hiszen"     "hogy"       "hogyan"     "igen"       "így"        "illetve"    "ill."       "ill"        "ilyen"      "ilyenkor"
 [89] "ison"       "ismét"      "itt"        "jó"         "jól"        "jobban"     "kell"       "kellett"    "keresztül"  "keressünk"  "ki"
[100] "kívül"      "között"     "közül"      "legalább"   "lehet"      "lehetett"   "legyen"     "lenne"      "lenni"      "lesz"       "lett"
[111] "maga"       "magát"      "majd"       "majd"       "már"        "más"        "másik"      "meg"        "még"        "mellett"    "mert"
[122] "mely"       "melyek"     "mi"         "mit"        "míg"        "miért"      "milyen"     "mikor"      "minden"     "mindent"    "mindenki"
[133] "mindig"     "mint"       "mintha"     "mivel"      "most"       "nagy"       "nagyobb"    "nagyon"     "ne"         "néha"       "nekem"
[144] "neki"       "nem"        "néhány"     "nélkül"     "nincs"      "olyan"      "ott"        "össze"      "ő"          "ők"         "őket"
[155] "pedig"      "persze"     "rá"         "s"          "saját"      "sem"        "semmi"      "sok"        "sokat"      "sokkal"     "számára"
[166] "szemben"    "szerint"    "szinte"     "talán"      "tehát"      "teljes"     "tovább"     "továbbá"    "több"       "úgy"        "ugyanis"
[177] "új"         "újabb"      "újra"       "után"       "utána"      "utolsó"     "vagy"       "vagyis"     "valaki"     "valami"     "valamint"
[188] "való"       "vagyok"     "van"        "vannak"     "volt"       "voltam"     "voltak"     "voltunk"    "vissza"     "vele"       "viszont"
[199] "volna"
```

3.13. ábra: Gyakori magyar kötőszavak

Ezeknek a lépéseknek hála a végső DFM-ben a tokenek (és oszlopok) száma 3041-ről 2520-ra csökkent, így a redundancia mérséklésével javítva a különböző algoritmusok hatékonyságát és futási idejét. Generáltam egy szófelhőt a DFM leggyakoribb kifejezéseiből, amelyek legalább 20-szor előfordultak a korpuszban. Ez a 3.14. ábrán látható, ahol az egyes szavak mérete az említések számával függ össze. A három leggyakrabban előforduló szó a külső behatású kár (gewaltschad), felújít (erneuert) és véghez vitt (durchgeführt).

A létrehozott DFM-ből egy új adathalmazt is kreáltam, ahol az egyes tokenek mellett a számosságuk szerepelt, vagyis az, hogy hányszor található meg a korpuszban.

Ebből az új adathalmazból látni lehetett, hogy a 2520 token jelentős többsége csak egy említéssel rendelkezik. Az elkopott kereket cseréjét tartalmazó jelentések beazonosításához 4 kulcsszó szinonimáit gyűjtöttem ki a leggyakoribb szavak közül: kerék (35 szinonima), elülső (8 szinonima), hátulsó (35 szinonima), csere (19 szinonima). A szinonimák viszonylag nagy száma annak is köszönhető, hogy a szavak szótőre visszabontása nem volt tökéletes. A kigyűjtött szinonimák segítségével generáltam egy második DFM-et is, amely rendelkezett 4 extra oszloppal (tire, front, back, replace). Ezek az oszlopok az összes szinonima szó együttes előfordulási számát tartalmazzák. A 3.15. ábrának megfelelően hozzárendeltem ezeket az oszlopokat a szervíz jelentésekhez, így képes voltam egyszerűen kiválasztani azokat a szerviz jelentéseket, amelyekben legalább egyszer szerepel a kerék és a csere szó egyszerre, valamilyen formában.

3.14. ábra: Szófelhő minimum 20 előfordulással rendelkező szavakkal

	serial_number	year_of_production	service_type	city_of_usage	service_comments	start_of_service	end_of_service	text_length	TIRE	FRONT	BACK	REPLACE
text1	516211B01298	2011	661	Neuenburg	Fahrzeugübergabe , Ladegerät in betrieb genommen ...	2012-01-26	2012-01-26	134	0	0	0	0
text2	516211B01298	2011	633	Neuenburg	Terminal motage durchgeführt .	2012-01-18	2012-01-18	30	0	0	0	0
text3	516211B01298	2011	617	Neuenburg	Servicebericht über Stornofunktion angelegt. (SB auf ...	2012-07-24	2012-07-24	61	0	0	0	0
text4	516211B01298	2011	617	Neuenburg	Hydraulikschlauch am Hubmast seitlich durchgesch...	2012-09-11	2012-09-13	110	0	0	0	0
text5	516211B01298	2011	617	Neuenburg	Durchführung der turnusmäßigen Wartungsarbeiten ...	2013-01-07	2013-01-07	78	0	0	0	0
text6	516211B01298	2011	617	Neuenburg	Wartung nach 1000 Betriebsstunden	2013-06-12	2013-06-12	35	0	0	0	0
text7	516211B01298	2011	617	Neuenburg	Batterie undicht , Aquamatik instandgesetzt , Batter...	2013-08-19	2013-08-19	135	0	0	0	0
text8	516211B01298	2011	617	Neuenburg	Räder auf der Lenkachse erneuert .	2013-11-28	2013-11-28	34	1	0	1	1

3.15. ábra: A szervíz jelentések első 8 eleme az új változókkal

3.4.2 Releváns szerviz jelentések beazonosítása

A négy új előállított változót (3.15. ábra) használtam a releváns szerviz jelentések megtalálására. **A 2 495 jelentés közül 158-nál volt a gumi szó** (vagy valamely feltérképezett szinonimája) **jelen**, **továbbá 117 jelentésnél volt a gumi és a csere szó egyszerre jelen** (TIRE > 0 és REPLACE > 0 feltétel). Ennél a mennyiségnél még úgy döntöttem a legnagyobb pontosságot eredményező módszer az, ha manuálisan ellenőrzöm a 158-ra leszűrt jelentések tartalmát, és elhagyom azokat a bejegyzéseket, ahol nem elkopás miatt történt gumicsere. Nagyobb adathalmaz esetében – ahol a manuális ellenőrzés nem lett volna járható út – szükség lett volna n-gram-ok használatára. N-gram-nak nevezzük számítógépes nyelvészetben az *n darab egységből* álló folytonos (egymást követő) sorozatokat tetszőleges szöveg vagy beszéd esetében. Az egység az lehet betű, szótag, szó, de akár még nagyobb szöveg elemek is. A mi esetünkben például, ha bigramokat (n = 2) használva feltérképeztük volna a szópárokat is, akkor könnyen szűrhettünk volna németül a „kerék csere” szópárra. N-gram-ok alkalmazása azonban drasztikusan megnöveli a változók (DFM oszlopainak) számát, ezért a gyakorlatban n értékére nem jellemző 3-nál nagyobb érték. A 158 jelentés manuális ellenőrzése alatt három problémás esetet azonosítottam be:

1. A kerék valamilyen formában említve van a jelentésben, de a tényleges probléma nem áll közvetlen kapcsolatban kerékkel (pl.: a kerék mögötti farlemez sérült).
2. Defekt vagy egyéb külső sérülés hatására történt a gumicsere.
3. A szerviz jelentése csak megállapítja, hogy elkopott a gumi a targoncán, de a csere nem történt meg. Ilyenkor általában az új kerekek megrendelésére kerül csak sor, majd kis idő elteltével (pár nap vagy egy hét) kerül sor a tényleges kerék cserére, amelynek szintén lesz saját szerviz jelentése.

A második és főleg a harmadik problémás eset beazonosításának megkönnyítésére készítettem hisztogramokat is a 158 szerviz időpontjáról, targoncákra

lebontva. A hisztogramok osztályait a dátumok adták egy hónapos bontásban. Ha egy targoncának egy hónapon belül egynél többször volt kerékkel kapcsolatos szervize, az mindenképp problémás esetre utal.

A kézi szűrés eredményeként 128 szerviz jelentésnél állapítottam meg, hogy a gumi elkopása miatt volt szükség a kerék cseréjére. A kerékcsere dátuma ismert minden esetben, de a regressziós modell eredményváltozójának (ami a gumi használatának a mennyisége) előállításához szükségünk van arra az információra is, hogy mikor kezdték el a lecserélt gumit használni. Az adathalmaz rendelkezett egy „munkaóra" oszloppal is, amely egy az egyben használható lett volna az eredményváltozóként, de sajnos minden szerviz jelentésnél NA (not assigned) értékkel rendelkezett. Ehelyett tehát minden bejegyzésnél megnéztem ismert-e egy korábbi kerékcserés szerviz időpontja, amelynek a kiváltó oka lehetett akár a gumi sérülése is. **Összesen 77 esetben tudtam ezzel a módszerrel meghatározni a gumi használatának a kezdetét. A szervizelés dátumából kivonva a gumi használat kezdetének dátumát azt is megkaptam, hogy a gumik cseréjéig hány nap telt el. Ez a változó szolgál a regressziós modell eredményváltozójaként.**

Ennek a változónak a minimuma 37 nap, átlaga 290 nap, míg a maximuma 807 nap volt. A változó nagy szórására a regressziós modell szempontjából vannak problémát nem jelentő magyarázatok és sajnos problémás magyarázatok is. A problémát nem jelentő magyarázatok közé tartozik a targoncák eltérő kihasználtsága, valamint a vezetési profilok különböző összetétele (pl. egy targonca használata során sokkal több időt tölt a gyors vezetési profilban az átlagoshoz képest). Ezek nem jelentenek problémát, hiszen rendelkezünk adatokkal róluk. Jelenleg azonban nem rendelkezünk adatokkal a targoncák környezetéről, hogy milyen körülmények között működnek, amelynek szintén hatása van a gumi kopásának sebességére. Sajnos a szervizelések során a gumivastagság sem került rögzítésre. A lecserélt kerekek állapotai között is nagy különbségek lehetnek. Előfordulhat például, hogy egy viszonylag új állapotú kereket is lecserélnek, ha a teljes flottán egyszerre akar a bérlő cég kereket cserélni. Ennél még valószínűbb eset az, hogy a kerekeket már hónapokkal korábban le kellett volna cserélni, mert annyira kopott állapotban volt, de a targoncát használó cégnél nem figyeltek erre.

Fontos tanulság, hogy a szerviz-naplózások részletezettsége különös jelentőséggel bír, és erre nyomatékosan fel kell hívni a szervizesek figyelmét. Ez az adatok címkézettségének hiányával felérő, általános probléma.

3.5 Targonca kerék élettartamának regressziós becslése

3.5.1 Az alkalmazott gépi tanulási technikák

Az adathalmaz felosztása tanító és teszt halmazokra bevett szokás az olyan gépi tanulási feladatoknál, ahol rendelkezünk címkézett adatokkal (osztályozás, regresszió). Ez azért van, mert a tanítás során fel nem használt teszt adatokkal becsülhetjük, hogy a modellünk hogyan fog teljesíteni a valós új adatokon. Ez az egyik legalapvetőbb módszer az overfitting[1] elleni védekezésre. Könnyen előfordulhat például, hogy egy osztályozási probléma megoldásához két különböző modellt is kipróbálunk. Az első 98%-os pontosságot ért el a tanító adatokon, míg a második csak 90%-ot. Megnézzük a modellek teljesítőképességét a teszt adatokon is, és azt tapasztaljuk, hogy az első modell csak 75%-ot ért el, míg a második modell 85%-ot. A második modell sokkal jobban használható, mert csak az adatok általános jellemzőit tanulta meg.

Fontos, hogy a tanító és teszt adathalmazok jól reprezentálják az eredeti adathalmazt. Ez osztályozási problémánál azt jelenti, hogy az osztályok aránya nem változik egyik részhalmazban sem, míg regressziónál az eredményváltozó eloszlásának kell minél jobban hasonlítani az eredeti állapothoz. Tipikusan az adatok 70-80%-át szokták tanító adatnak használni. Főleg deep learning alkalmazásoknál jellemző [23] egy harmadik részhalmaz, a validációs adathalmaz használata is. Neurális hálók tanítása során minden egyes tanítási fázis után megnézik a hiba mértékét a validációs adathalmaz segítségével is, és a neurális háló tanítását akkor állítják le, amikor a validációs hiba mértéke tetszőleges számú tanítási fázis után sem javult tovább (miközben a tanító adathalmazon a hiba mértéke tovább csökkenhetett, mert a modell elkezdett rátanulni csak a tanító adathalmazra jellemző tulajdonságokra, természetes zajokra). A tanítás leállítása után a legjobb validációs hibát produkáló modellel kell felhasználni a teszt adathalmazt a végső kiértékeléshez. Jellemző felosztási arány ilyen esetekben [23] a 70% tanító adat, 20% validációs adat és 10% teszt adat.

A validációs adathalmaz használatának egy tovább gondolt módja a *cross-validation* (CV) [24] technikája, amelyet előszeretettel alkalmaznak hagyományos gépi tanulási problémáknál kis és közepes méretű adathalmazok esetén. Több típusa is van, de az általam is használt k-fold cross-validation lényege a következő: a tanító adathalmazt felosztjuk k darab részre (figyelembe véve, hogy az egyes részek jól reprezentálják a teljes adathalmazt), majd az egyik részt félre teszi az algoritmus validációs

adathalmaznak. A cross-validation k ilyen iteráció után fejeződik be, ahol mindegyik *fold* pontosan egyszer volt validációs adathalmazként használva. A k db eredmény átlagolásra kerül a pontosság vagy egyéb metrika becslésére. A k-fold módszer előnye, hogy a tanító adathalmaz minden pontja felhasználásra kerül tanításra és validációra is. Számítás intenzív módszerről van szó, hiszen k-szoros tanítást végzünk. Legnagyobb haszna a CV módszernek kis adathalmazok esetén van (mint például a mi esetünkben, ahol összesen 77 bejegyzés van), mivel használatával nincs feltétlen szükség külön teszt adathalmaz készítésére, így a modell több pontból tanulhat. Minden regressziós modellemhez tízszer megismételt 3-fold CV-t használtam (összesen tehát 30 tanítás kerül átlagolásra modellenként az overfitting elkerülése érdekében), amely egy jól bevált érték k-fold CV esetében kis méretű adathalmazokra.

Manapság a legsikeresebb predikciós algoritmusok alapja az *ensemble learning* (együttes tanulás). Ezek az algoritmusok sok modell együttes figyelembe vételével érnek el magasabb pontosságot. Ez döntési fákra vetítve azt jelenti, hogy egyszerre sok döntési fának a predikcióját vesszük figyelembe. Ezután osztályozási problémánál többségi szabály alapján dönthetünk a legvalószínűbb osztály javára, regressziós problémáknál pedig a predikciók súlyozott átlagát lehet venni. Az átlagolásnak hála az egyes modellek (fák) eltérő hibái kiátlagolódnak. Az *ensemble learning* módszerek közül az egyik legismertebb a Random forest (RF) algoritmus [25]. Mind osztályozási és regressziós problémákra széles körben sikeresen alkalmazzák. Az algoritmusnak bemeneti paraméterként lehet megadni, hogy hány döntési fát építsen. A fák között a különbséget az algoritmus kétféleképpen garantálja:

- Minden fa építéséhez az adatok egy véletlen mintavételezésű részhalmazát használja csak. A mintavételezés visszatétellel történik, tehát egy bejegyzés kétszer is felhasználásra kerülhet, míg más bejegyzések akár soha nem.
- Egy *node* szétválasztásánál a változók csak egy véletlenszerű részhalmazát veszi figyelembe. Ez a RF algoritmus újítása, amelyről a nevét is kapta.

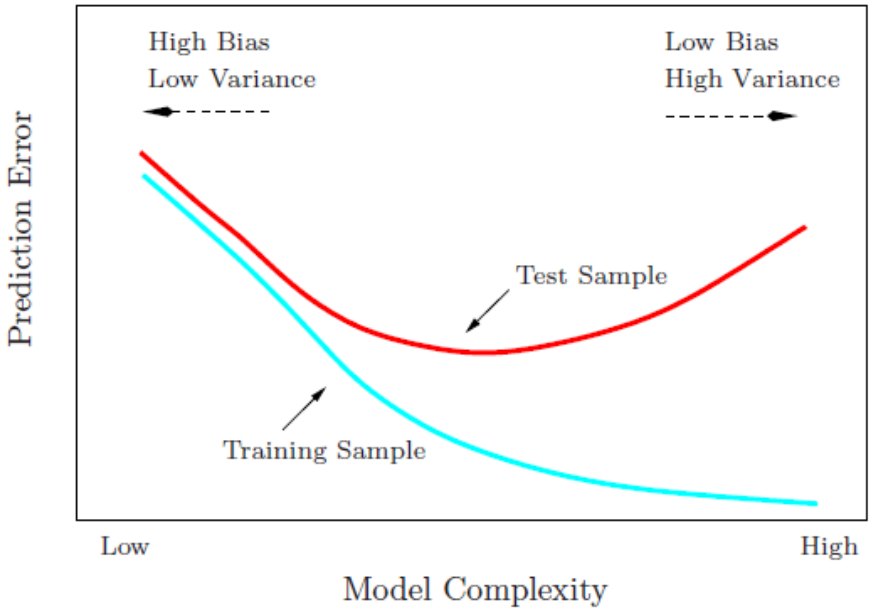
A döntési fák különbözőségéből adódik, hogy az egyes fák hibája jól kiátlagolható lesz. A hiba komponensnek három összetevője van:

- Bias: a hiba azon komponense, ami abból ered, hogy a modell nem képes jól reprezentálni az adatok tényleges struktúráját (az adatok „függvényét” nem

képes leírni). Gyakran hívják az underfitting komponensnek, hiszen a mértéke a túl egyszerű modelleknél számottevő.

- Variancia: mértéke azt írja le, hogy mennyire instabil a modell, mennyire ad eltérő predikciót egy bizonyos pontra, ha más tanuló adatokkal tanították (az új adatoknak természetesen ugyanabból a fizikai folyamatból kell származnia). A bias értékével ellentétben ennek a hibakomponensnek az értéke pont a komplex modellek esetén számottevő (ábra), ezért overfitting komponensnek is hívják. Az *ensemble* módszerek igyekeznek az egyik komponens magas értékét kiküszöbölni úgy, hogy közben nem rontanak sokat a másik komponens értékén.
- A nem csökkenthető hiba: az adat valós függvényébe beleszövődő véletlenszerű hiba, amelyet a modell nem tud megtanulni.

Az RF algoritmus sok szintből álló, mély fákat céloz építeni, amelyeknek így a bias értéke alacsony, varianciájuk pedig magas. A varianciát az egyes fák átlagolásával csökkenti.



3.16. ábra: Modell komplexitása és a predikciós hiba kapcsolata [26]

Egy másik, egyre népszerűbb *ensemble tree* algoritmus a Gradient Boosted Decision Trees [27] (GBDT). Ez az algoritmus szekvenciálisan épít sok, alacsony komplexitású fát. Az éppen aktuálisan épülő fa mindig az előző fa hibáit igyekszik kijavítani, ezért a hibásan jósolt bejegyzések nagyobb súlyt kapnak. Az új fa ezáltal jó

eséllyel képes lesz jó becslést adni az eddig hibásan becsült bejegyzésekre, miközben korábban jól becsült bejegyzéseken romolhat a hatásfoka. A fák alacsony komplexitása miatt a variancia alapból kicsi, és mivel az egyes fák az előző fák hibáiból tanulnak, ezért összegzésükből egy komplex modell állítható össze. Ez a modell képes jól lefedni az adathalmaz értéktartományát jó becslést adó fákkal, így a bias is kicsi lesz.

Mind az RF és GBDT algoritmus pozitív tulajdonsága, hogy nem érzékenyek a változók korreláltságára, valamint az eltérő változó típusokat is jól kezelik (numerikus, típus változó). Sajnos *ensemble* (együttes) jellegük miatt mindkettőnél olyan, mintha a becslések csak egy fekete dobozból jönnének. Nincs egy egyszerű képlet vagy döntési fa, amely jól leírja a modell teljes működését, mint más, egyszerűbb algoritmusok esetében. Ennek következménye, hogy az új bemeneti adatokat ugyanarra az alakra kell hozni (számított változók előállítása, átskálázások stb.), amely alakon a modell tanítása is történt. Ez komplex modellek esetében nem mindig egyszerű feladat (pl. szöveg analízis során, ahol a predikciót végző algoritmusok bemenete erősen absztrakt jelentésű, nagy méretű mátrixok). A GBDT fa építési fázisa nem párhuzamosítható (csak a *cross-validation* folyamata), mint az RF-nél, valamint a bemeneti paraméterei is alaposabb hangolást igényelnek a jó pontosság eléréséhez. A GBDT tanítása fázisa lassabb, de cserébe a modell tárolásához kevesebb memória szükséges, és gyorsabban képes predikciót hozni. A szakirodalmak szerint [28] [29] [30] a GBDT a legjobb pontosságot biztosító *off-the-shelf* algoritmus, ha a paraméterek hangolását megfelelően elvégezzük, így a két algoritmus közül a GBDT-re esett a választásom.

3.5.2 Az adathalmaz előkészítése

A *fleetmanager* adathalmaz aggregált CAN busz adatokat tartalmaz 69 targoncáról. A bejegyzések olyan változókat tartalmaznak, mint például a megtett távolság vagy a vezetéssel és emeléssel töltött idő. A bejegyzéseket vezetési profilok szerint csoportosítottam, hogy további prediktív bemeneti változókat nyerjek ki az adathalmazból a regressziós modell számára. A másik adathalmaz targoncák több évnyi szerviz jelentéseiből állt, amely a szerviz dátuma mellett a szerviz során megállapított hibákat és elvégzett javításokat leíró szöveges változót is tartalmazott. Ezen változó segítségével beazonosítottam 128 olyan bejegyzést, ahol a gumi elkopása miatt történt kerékcsere a 69 targonca egyikén. Ezekből a bejegyzésekből 77-nél sikerült megállapítanom korábbi szerviz jelentéseknek hála, hogy a gumit mikor kezdték el

használni. Így a regressziós modell számára 77 darab eredményváltozóval rendelkezem. Az eredményváltozó nem más, mint a gumi élettartama napokban kifejezve.

A különböző paraméterekkel rendelkező regressziós modellek kipróbálása előtt szükség van a magyarázó változók és az eredmény változó összepárosítására. Mivel a bemeneti változók csak 10 perces intervallumra vonatkoznak, a magyarázó változó pedig több hetet vagy akár hónapokat is lefed (attól függően, hogy mennyi idő alatt kopott el a gumi), ezért szükség van a fleetmanager adatok aggregálására. Az előkészítési folyamat lépései láthatók a 3.17. ábrán.

3.17. ábra: Az adathalmazok előkészítése a regresszióra

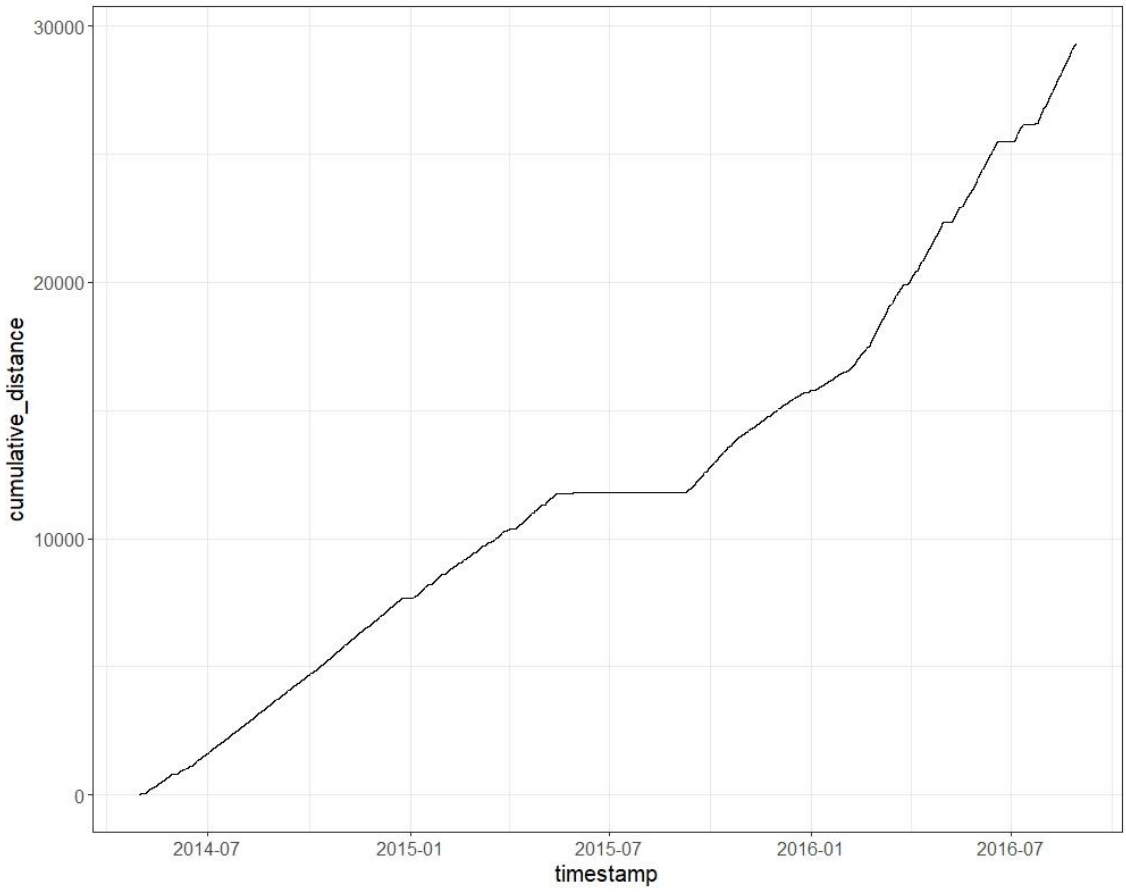
Mindezek előtt elvégeztem az adathalmazon ugyanazokat a lépéseket, amelyet a klaszterezés során is tettem, hogy megkapjam klaszterezéshez használt változókat. Beolvastam az elmentett klaszter centereket, majd euklideszi távolságot számolva a bejegyzések és a középpontok között, mindegyik bejegyzéshez hozzárendeltem egy vezetési profilt. Ez a módszer sikeresnek bizonyult, mivel szinte majdnem pontosan ugyanazt a klaszter eloszlást kaptam, mint amit a csoportosításkor.

Sajnos a szerviz jelentések és a fleetmanager adatok által lefedett időintervallumok között az átfedés nem volt megfelelő a magyarázó változók aggregálásához. A tipikus eset az volt, hogy a szervizelések korábban történtek meg, mint az adott targoncához tartozó legkorábbi fleetmanager bejegyzés. A két adathalmaz közötti átfedés mértékét egy egyszerű változó segítségével ellenőriztem. Minden egyes szerviz jelentésnél az adott targonca fleetmanager bejegyzéseit szűrtem idő alapján a gumik elhasználódásának intervallumára. Azt tapasztaltam, hogy a 77 alkalomból 50-nél egyáltalán nem rendelkezek fleetmanager adatokkal az adott intervallumra.

Azonban mindegyik targoncához több ezer fleetmanager bejegyzéssel rendelkezek, ezért megvizsgáltam mennyire vinnék nagy hibát a modellbe, ha a fleetmanager változók összes értékét összegezném, és az alábbi szorzó értékével normalizálnám azokat:

$$Szorzó = \frac{SzervizDátuma - GumiKezdetiDátuma}{UtolsóFleetmanagerDátuma - ElsőFleetManagerDátuma}$$

Mivel a dátumok napra pontosan tartalmazták az időt, ezért egymásból kivonva őket a két dátum között eltelt napok számát kaptam meg. A számlálóban a regressziós modellhez használni kívánt eredményváltozó értéke található, és ezt osztom le a targoncához tartozó legkorábbi és legutolsó fleetmanager bejegyzés közt eltelt napok számával. A szorzó eredményes használatához az szükséges, hogy a bemeneti változók jellege hosszútávon ne változzon jelentősen.



3.18. ábra: A távolság változó összegzett értéke egy targoncára az idő függvényében

Ennek ellenőrzésére több targoncánál is ábrázoltam az idő függvényében a változók összegének az alakulását. A 3.18. ábra a legrosszabb esetet ábrázolja (ideális esetben egy teljesen lineáris függvényt látnánk), az összegzett távolság időbeli alakulását

annál a targoncánál, amelyhez a legtöbb – majd 80 ezer – fleetmanager bejegyzés tartozik. A vízszintes tengelyen láthatjuk, hogy 2 év alatt gyűjtött a targonca ennyi bejegyzést. Látható, hogy vannak hosszabb-rövidebb időszakok, ahol a targonca nem volt használatban, illetve a középső hosszú kihagyás után a targoncát intenzívebben kezdték el használni, amelynek az oka a többi változó diagramját is figyelembe véve a hosszabb műszak volt. Mivel a legrosszabb esetben is lineáris jelleget mutat az összegzett változó értéke, valamint az adathalmazok rossz átfedéséből adódó nagy mennyiségű hiányzó adat miatt nem láttam jobb lehetőséget, ezért a szorzó használata mellett döntöttem.

A fleetmanager adatok összegzése és a szorzóval való normalizálás után az alábbi bemeneti változók álltak elő a regressziós modell számára:

- ***profile1_ratio***: a targonca által az első vezetési profilban töltött idő aránya.
- ***profile2_ratio***: a targonca által a második vezetési profilban töltött idő aránya.
- ***profile1_hours***: a targonca által az első vezetési profilban töltött összes órák száma, a szorzóval normalizálva.
- ***profile2_hours***: a targonca által a második vezetési profilban töltött összes órák száma, a szorzóval normalizálva.
- ***cumulative_drive_time***: a targonca által vezetéssel töltött összes idő (emelés nélkül), a szorzóval normalizálva.
- ***cumulative_lift_and_drive_time***: a targonca által vezetéssel töltött összes idő, miközben tehert is cipelt a villáján, a szorzóval normalizálva.
- ***cumulative_distance***: targonca által megtett összes távolság, a szorzóval normalizálva.

A felsorolásban lévő változók mellett nem éreztem szükségesnek a harmadik vezetési profilhoz is hasonló változók létrehozását, főleg a ***profile3_ratio*** lett volna teljesen redundáns (hiszen a három vezetési profil arányának összege 1), valamint a legtöbb targoncánál messze ez volt messze a legkevésbé jellemző vezetési profil. A klaszterezéssel ellentétben a ***lift_only_time*** változó értékei se kerültek felhasználásra, hiszen ezekben az időszakokban a targonca nincs mozgásban, így a gumik felületei se koptak.

3.5.3 A kiértékelési tanulságok

A kipróbált regressziós modellek tehát mindegyike a Gradient Boosted Decision Tree algoritmust használta, 10-szer megismételt 3-fold cross-validationt használva, amely ideális a mindössze 77 bejegyzésből álló adathalmazunkhoz.

A modellek három tényezőben tértek el egymástól. Egyrészt a használt bemeneti változókban. Volt olyan modell is például, amelyhez hozzávettem az előző fejezetben említett változók mellett azokat a faktoriális változókat is, hogy melyik városban, és melyik cég használja a targoncát, de ez (ilyen kevés adattal) nem bizonyult hasznosnak. Emellett kísérleteztem olyan kombinációkkal is, ahol csak 2-4 korrelálatlan bemeneti változót adtam meg, de mivel a GBDT modell nem érzékeny erre, tapasztalataim szerint ez inkább a pontosság romlását eredményezte az információ vesztés enyhe mértéke miatt.

Megvizsgáltam a szokásos előfeldolgozási eljárások hatását (második különbség) is a modellek pontosságára. Ezen eljárások közé tartozik az adathalmaz felosztása tanító és teszt részhalmazokra, a változók átskálázása 0 középértékű, 1 szórású változókká, valamint a Yeo-Johnson transzformáció [31] is. E transzformáció célja a variancia stabilizálása azáltal, hogy a változók eloszlását közelíti a normális eloszláshoz. A GBDT rugalmasságának köszönhetően a transzformációknak nem volt számottevő jótékony hatása, viszont az adathalmazból teszt bejegyzések elkülönítése inkább negatív hatással bírt a modellek összteljesítményére, hiszen még kevesebb adatból tudtak tanulni. Ezért végül mind a 77 bejegyzésen tanultak a modellek, és a cross-validation során kiszámolt validációs pontosságot átlaga adta a modellek pontosságát.

A modellek közötti utolsó eltérés az a paraméterek lehetséges értékei között voltak. Vannak paraméterek, amelyeket konstans értéken tartottam minden modell számára, az eltérés az alábbi paraméterek között volt:

- ***Eta:*** a tanulási ráta, azaz az aktuális fa milyen mértékben igyekezzen kijavítani az előző fa téves predikcióit
- ***Nrounds:*** hány fából épüljön fel a modell
- ***Max_depth:*** a fák maximális mélysége, azaz a maximális döntések száma egy predikcióhoz
- ***Min_child_weight:*** a fában az elágazások létrejöttéhez szükséges minimum eltérés (a két levél között) értékét befolyásolja

- ***Colsample_by_tree:*** a változók hányad részét használja minden fa építésénél (oszlopok kiválasztása véletlenszerű, mint az RF algoritmusnál)

A fenti paraméterek finomhangolásához a *grid search* módszert alkalmaztam. Ez annyit jelent, hogy mindegyik paraméterhez megadtam milyen értékeket próbáljon ki, és az algoritmus a paraméterek összes kombinációját kipróbálja a modell alkotáshoz. Természetesen ez a módszer csak kis és közepes méretű adathalmazok esetén járható út, nagyobb adathalmazokra léteznek más megközelítések is. A 3 különböző paraméter-finomhangolási „rács", amit kipróbáltam az 4 320, 324 és 900 különböző paraméter kombinációval rendelkezett. Az algoritmus futási idejét még befolyásolja a cross-validation mértéke (30-as szorzó), az ideális paraméter kombinációban a fák száma (100 és 200 közötti szorzó), valamint maga a tanításhoz használt bejegyzések száma.

A modellek pontosságát a regressziónál gyakran használatos RMSE (Root Mean Square Error) mérőszám segítségével értékeltem ki. **A legkisebb elért RMSE érték csak 91 nap volt** (a mértékegysége ugyanaz, mint az eredményváltozóé) sajnos, amely az eredményváltozó medián középértékének a 35%-a. Ez közel sem elég jó pontosság, bár fontos megemlíteni, hogy az RMSE mérőszám különösen érzékeny a nagy hibázásokra a négyzetes összegzésnek köszönhetően. A legpontosabb becslés hibája 0 nap volt (becsült és valós érték is 168 nap volt), míg a legpontatlanabb becslés hibája 269 nap volt, ahol a 241 napos becsült érték helyett 511 nap volt a valóság**. A hibák átlaga 25,5 nap, amely az eredményváltozó medián értékének 9.8%-a.**

A pontatlanság fő oka szerintem a kevés tanuló adatban rejlik. Ezt látszik igazolni az a tény is, hogy ha az algoritmus futtatásához használt kódnál eltérő seed-számot adok meg a pszeudorandom számgenerátornak, akkor a predikciók jelentős mértékben módosulnak, amely a modell instabil jellegére utal.

A két kerékcsere dátuma közt eltelt napok száma használata, mint a regresszió eredményváltozója szintén problémás. Egyrészt a szervizjelentések és fleetmanager adatok átfedése nagyon gyenge volt. Ebből kifolyólag az extrapolálni kellett a fleetmanager adatokat, amely könnyen okozhatott számottevő hibát, hiszen voltak olyan targoncák, amelyek rendelkeztek kihasználatlan időszakokkal (lásd 3.18. ábra). Az eredményváltozó másik problémája a nagy szórása, amely csak részben az eltérő használati profilok eredménye. Mint ahogy korábban említettem már, a gumicserék nem

egységesen történtek egy bizonyos vastagság elérésekor, hanem egyszerűen akkor, amikor a bérlő cégnél úgy gondolták, hogy szervizbe küldik a targoncát.

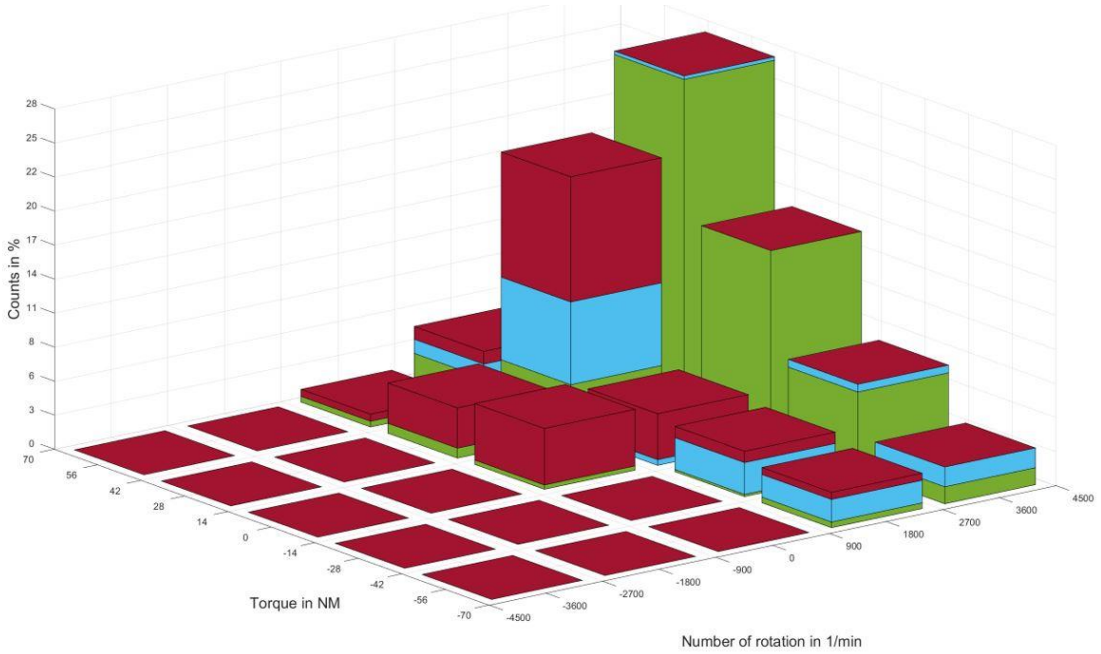
3.6 Tovább fejlesztési lehetőségek

A hatékony *supervised machine learninghez* elengedhetetlenek a jó minőségű tanító adatok. Bár targoncák gumi kopásához nem egyszerű releváns adatokat mérni és begyűjteni, a jövőben továbbra is együtt kell működnünk a STILL-lel, és az ebben a dolgozatban felvázolt problémákra megoldásokat kell találnunk. A legfőbb problémákra az alábbiak jelenthetnek megoldást:

- A historikus adatok gyűjtésének folytatása, és olyan szerviz jelentések és fleetmanager adathalmazok összeállítása, amelyek ugyanazt az időintervallumot fedik le teljes egészében (feltéve, ha tudjuk azt, hogy a fleetmanager adatok gyűjtésének kezdetével új gumival állt üzembe a targonca). Ez eliminálná a tökéletlen ***days_passed*** változó használatának szükségességét, hiszen a fleetmanager bejegyzések ***readout_duration*** változójának összegzésével pontosan tudnánk mennyit volt használatban a targonca. Szintén fontos lenne minél több targoncát bevonni az adatgyűjtésbe. A további javaslatok együttes alkalmazásával véleményem szerint a mostani 77 bejegyzés 4-5-szöröse már elegendő lehet egy olyan modell megalkotásához, amely képes valódi hasznot biztosítani a pontos előrejelzésekkel.
- A jobb predikciók érdekében fontos lehet további mennyiségek mérése is. Ez egyrészt történhet a targonca MCU egységeinek a módosításával, hogy a CAN busz adatokból több dolog legyen elmentve (például az, hogy mekkora súlyt cipel a targonca, mekkora távolságot tett meg eközben, vagy például az átlagos forgatónyomaték a mérési időintervallum alatt). Másrészt történhet egyedi, FPGA alapú mérőberendezések fejlesztésével, amely eszköz szenzorok segítségével képes lenne környezeti változók mérésére (hőmérséklet, páratartalom stb.). Egy ilyen eszköz fejlesztése már folyamatban van [32].
- Szervizelés során, ha történik gumi csere, akkor fontos lenne a lecserélt gumik vastagságának rögzítése. Emellett fontos lenne tudni a kerék típusát, és pontos dimenzióit újszerű állapotában. Egy olyan adathalmaz elkészítése

is rendkívül hasznos lehetne, amelyben mondjuk 100 munkaóránként rögzítésre kerülnének a gumik vastagsága (a targonca mind a 4 gumijához külön-külön) eltérő környezetben dolgozó targoncáknál.

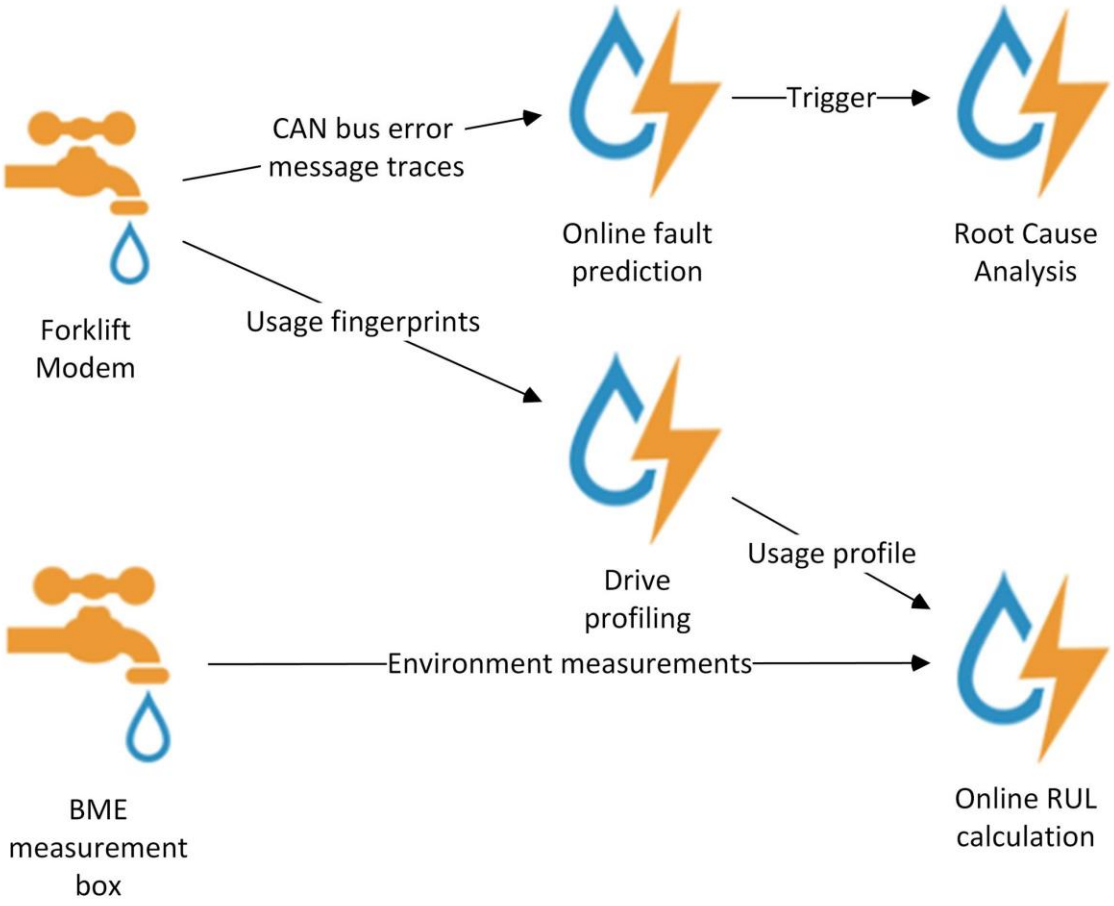
A rendelkezésre álló CAN busz adatokat is fel lehet használni a jövőben. A STILL szerint a CAN busz adatok segítségével készíthetők 2 dimenziós hisztogramok (fingerprinteknek nevezik). Egy *fingerprint* 10 percnyi CAN busz adat aggregálásából készülne, és a hisztogramok minden változó páros esetében 5x5 (tehát összesen 25 darab) oszlopból állna a javaslatuk szerint. A 3.19. ábrán látható egy példa fingerprint, amelynél a két változó a forgatónyomaték és a fordulatszám. Jelenleg vizsgáljuk egy RUL modell készítésének a lehetőségét, ezen fingerprint adatok alapján. Az elképzelés szerint a 10 percenként generálódó fingerprinteket a targoncán található GSM modem vektorokba tömörítve beküldhetné (ahol a vektorban lévő számok a hisztogram oszlopok magasságát adná meg). A felhőbeli feldolgozó egység pedig a 10 percenként érkező fingerprint csomagokból eleget begyűjtve képes lenne a targonca RUL értékét frissíteni adott időközönként.



3.19. ábra: Példa egy CAN busz fingerprintre

A felhőbeli feldolgozó egység váza látható a 3.20. ábrán, amely megvalósítása Apache Storm valósidejű feldolgozó (stream processor) keretrendszer [33] segítségével fog megtörténni. Az adatok forrása a targoncákon már jelenleg is megtalálható GSM modemek (amelyek kevésbé költséghatékonyak a szokásos IoT eszközökhöz képest az

előfizetői díjak miatt), valamint az általunk fejlesztett FPGA alapú mérőberendezés [32] lesznek. A mérőberendezés fogja biztosítani a környezeti változók értékeit, míg a modem fogja előállítani a fingerprinteket a gumik élettartamának becsléséhez. A CAN busz hibaüzenetek is felhasználásra fognak kerülni RCA megvalósításának érdekében. Célunk mintázatfelismerést alkalmazva, a hibaüzenetekből a meghibásodások bekövetkezését megjósolni, valamint az okukat meghatározni.



3.20. ábra: A felhőbeli feldolgozás vázlata

4 Összefoglalás

A dolgozat elején bemutattam, hogy a kiberfizikai rendszerek hogyan képesek segíteni az ipari automatizálás továbbfejlesztésében. A realizálásához olyan koncepciók alkalmazása szükséges, mint a SOA vagy az edge computing, valamint szükséges a különböző rendszerek között egy egységes kommunikációs szintaxis megteremtése is, erre a MIMOSA szabvány próbál megoldást nyújtani.

Röviden bemutattam a MANTIS projektet, amelynek célja proaktív karbantartási rendszerek megvalósítása már létező technológiák segítségével (szenzorok, FPGA-k, cloud computing stb.). A projektben számos nagyvállalat is részt vesz, mivel a hagyományos karbantartási eljárások nem hatékonyak sem költségek, sem produktivitás szempontjából. Az eszközök és rendszerek karbantartásának ezen újszerű megközelítése kihasználja az ipari környezetben rendelkezésre álló – de többnyire kihasználatlan – nagy adatmennyiséget. A projekt egyik ipari együttműködő partnere a STILL nemzetközi vállalat, amely az általa gyártott és kibérelt targoncák karbantartását szeretné hatékonyabbá tenni. Együttműködésünk során az első közös célunk egy olyan rendszer megvalósítása, amely képes a targonca kerekeinek (a gumi felületek) élettartamának becslésére. Ennek megvalósításához nyers CAN busz adatokat, a targoncák vezérlő egysége által gyűjtött – úgynevezett *fleetmanager* – adatokat, valamint szervizelési jelentések adathalmazát nyújtotta a STILL számunkra.

Megterveztem és implementáltam egy módszert arra, hogy a fleetmanager bejegyzéseket három különböző vezetési profilba lehessen csoportosítani. A STILL szakértői szerint ezek a vezetési profilok különböző mértékű terhelést jelentést jelentenek a gumik számára, ezért beazonosításuk hasznos a gumi élettartamának becslésére. A vezetési profilokat olyan változók segítenek beazonosítani, mint például az átlagos sebesség, vagy az irányváltások száma. Az adatok pontos csoportosításához egy centroid-alapú és egy hierachikus klaszterező algoritmust együttesen alkalmaztam. A klaszterezés eredményeit bemeneti változók előállítására használtam fel a regressziós modell számára, amely modell feladata a Remaining Useful Life (RUL) becslése.

A szöveges, német nyelvű szerviz jelentések feldolgozását is megvalósítottam. Itt a cél a több százezer jelentés közül azok beazonosítása volt, ahol a gumik elkopása miatt kerékcsere történt azokon a targoncákon, amelyekhez fleetmanager adatok is elérhetőek.

128 ilyen szerviz jelentés volt az adathalmazban, amelyek közül 77-hez sikerült meghatározni az elkopott gumi használatának a kezdetét a rendelkezésre álló adatokból. Ismerve azt, hogy a gumit mikor kezdték el használni, majd mikor cserélték le, elő tudtam állítani a regressziós modell eredmény változóját, a gumi élettartamát napokban kifejezve. A regressziós modell feladata tehát a szerviz jelentésekből kinyert gumi élettartam változó becslése, a fleetmanager adathalmazból előállított paraméterek segítségével.

A regressziós modellhez a kevés tanuló adatból kifolyólag egy olyan gyakran használatos *ensamble tree* alapú algoritmust használata mellett döntöttem, amely alapos (és számítás igényes) paraméter optimalizáció mellett képes jobb becsléseket adni sok más hagyományos módszernél. A modell túltanulása elleni védekezés érdekében és a kiértékeléshez a *cross-validation* (keresztvalidáció) technikát alkalmaztam. A regresszió során kapott eredmények többnyire nem voltak kielégítőek. Ennek az okait ismertettem, valamint a lehetséges megoldásokat is felsoroltam. A megvalósításukhoz a továbbiakban is szorosan együtt kell majd működnünk a STILL-lel.

Tudomásom szerint a haszongépjárművek ilyen proaktív karbantartási feladatokra alkalmazott adatgyűjtési és feldolgozási módszertana - különösen ilyen átfogó módon - még nem került publikálásra. Dolgozatom ilyen szempontból is egyedinek és újszerűnek tekinthető.

Irodalomjegyzék

[1] A MANTIS projekt oldala: http://www.mantis-project.eu/ (2017.10.01.)

[2] Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016 (Online: 2017 szept.) http://www.gartner.com/newsroom/id/3598917

[3] B. X. Huang: *Cyber physical systems: a survey*, 2008

[4] E. Jantunen, U. Zurutuza, L.L. Ferreira, P. Varga: *Optimising Maintanance: What are the expectations for Cyber Physical Systems*, 2016

[5] T. Erl: *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005

[6] W3C: *Web Services Activity*, 2002. Elérhető: http://www.w3.org/2002/ws/

[7] A. I. Omer, M. M. Taleb: *Architecture of Industrial Automation Systems*. European Scientific Journal, ESJ 10.3, 2014

[8] E. Jantunen, U. Gorostegi, U. Zurutuza, F. Larrinaga, M. Albano, G. Di Orio, P. Maló, Cs. Hegedűs: *The Way Cyber Physical Systems Will Revolutionise Maintenance*, 2017

[9] MQTT protokoll hivatalos oldala: http://mqtt.org/ (2017.10.01.)

[10] MIMOSA hivatalos oldala: http://www.mimosa.org/mimosa/ (2017.10.01.)

[11] L. L. Ferreira, M. Albano, J. Silva, D. Martinho, G. Marreiros, G. Di Orio, P. Maló, H. Ferreira: *A Pilot for Proactive Maintenance in Industry 4.0*, 2017

[12] G. Di Orio, P. Malo, M. Albano, L. L. Ferreira, Cs. Hegedűs, P. Varga, I. Moldován: *Interoperable and Interconnected CPS-populated Systems for Proactive Maintenance*, 2017

[13] Az Arrowhead projekt oldala: http://www.arrowhead.eu/ (2017.10.01.)

[14] Cs. Hegedűs: *Cyber-physical systems collaborating in a service oriented architecture*, 2016

[15] RX 20-as targonca adatlapja: https://www.still.de/en-DE/trucks/new-trucks/electric-forklift-trucks/rx-20-14-20-t.html (2017.01.01.)

[16] Hodeghatta U.R., Nayak U.: *Unsupervised Machine Learning*. In: *Business Analytics Using R - A Practical Approach*. Apress, Berkeley, CA. 2017

[17] S.P. Lloyd: *Least squares quantization in PCM*. IEEE Transactions on Information Theory, 28.2: 129-137, 1982

[18] B. Desgraupes: *Clustering Indices*, 2013. Elérhető: https://cran.r-project.org/web/packages/clusterCrit/vignettes/clusterCrit.pdf

[19] D. Arthur, S. Vassilvitskii: *k-means++: The Advantages of Careful Seeding*. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035, 2007

[20] Klaszterezési példa adathalmaz forrása: http://scikit-learn.org/stable/modules/clustering.html

[21] J.H. Ward: *Hierarchical Grouping to Optimize an Objective Function*, Journal of the American Statistical Association. 58 (301): 236–244, 1963

[22] A szöveg analízist segítő quanteda R csomag honlapja (dokumentációval): http://quanteda.io/

[23] J. Kunce, S. Chatterjee: *A Machine-Learning Approach to Parameter Estimation*

[24] R. Kohavi: *A study of cross-validation and bootstrap for accuracy estimation and model selection*, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann. 2 (12): 1137–1143, 1995

[25] L. Breiman: *Random Forests*. Machine Learning (journal). 45 (1): 5–32, 2001

[26] S. Fortmann-Roe: *Understanding the Bias-Variance Tradeoff*, 2012 (Online: 2017 szept.) http://scott.fortmann-roe.com/docs/BiasVariance.html

[27] J.H. Friedman: *Stochastic Gradient Boosting*, March 1999

[28] R. Caruana, A. Niculescu-Mizil: *An Empirical Comparison of Supervised Learning Algorithms Using Different Performance Metrics*, 2005

[29] R. Caruana, N. Karampatziakis, A. Yessenalina*: An Empirical Evaluation of Supervised Learning in High Dimensions*, 2008

[30] M. Fernández-Delgado, E. Cernadas, S. Barro: *Do we need hundreds of classifiers to solve real world classification problems?*, Journal of Machine Learning Research 15: 3133-3181, 2014

[31] I.K. Yeo, R.A. Johnson: *A new family of power transformations to improve normality and symmetry*, Biometrika Volume 87, Issue 4, Pages 954-959, December 2000

[32] A. Frankó: *Information-driven proactive maintenance systems*, 2017 (Online: http://tdk.bme.hu/VIK/Info/Informaciovezerelt-proaktiv-karbantartasi)

[33] Apache Storm stream feldolgozó keretrendszer: http://storm.apache.org/ (2017.10.01.)