LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

AUTOVADĪTĀJA IDENTIFIKĀCIJA, IZMANTOJOT OBD2 DATUS

MAĢISTRA DARBS

Autors: Dmitrijs Žukovs
Stud. apl. Nr. dz06022
Darba vadītājs: profesors Kārlis Podnieks,
datorzinātnes maģistra studiju
programmas direktors

RĪGA 2019

ANOTĀCIJA

Mūsdienās attīstošās datizraces tehnoloģijas un metodes ir radījušas lielu interesi par dažādiem ievāktajiem datiem. Ar 1996. g. ASV un 2003. g. ES, visiem šajās teritorijas tirgotajiem transportlīdzekļiem jābūt aprīkotiem ar OBD2, kas ir sensoru datu apstrādes standarts. Iepriekš veiktie pētījumi ir pierādījuši, ka ir iespējams no šiem datiem identificēt braucēju, ja tie tiek vākti strikti kontrolētos eksperimentos ar noteiktiem maršrutiem un braukšanas laikiem. Šajā darbā tika apskatīts, vai ir iespējams identificēt braucēju no maza sensoru skaita OBD2 datiem, kas ir ievākti no 8 autobraucējiem vadot vienu transportlīdzekli dažādos apstākļos. Datu klasifikācijai tika izmantota programma WEKA un tajā pieejamie modeļi. Lai noteikt savākto parametru informatīvā ieguldījuma reitingu tika izmantots InfoGainAttributeEval atribūtu reitinga modelis. Izmantojot 6 augstāk novērtētos atribūtus un Random Forest algoritmu ir iespējams klasificēt braucēju ar 90% varbūtību. Papildus arī tiek savākti augstākas frekvences dati un parādīts, kā izmantojot neuzraudzīto metodi, iespējams identificēt autovadītāju skaitu no braukšanas datiem.

Atslēgvārdi: datizrace, OBD2, autovadītāja identifikācija, u-shapelet, laika rindas.

ABSTRACT

Driver identification using OBD2 data.

Nowadays, developing data mining technologies and methods have generated great interest in the various collected data. With 1996 USA and 2003 In the EU, all vehicles sold in the area must be equipped with OBD2, which is the standard for vehicles sensor data processing. Previous studies have shown that it is possible to identify a rider from these data if it is collected in strictly controlled experiments with certain routes and driving times. This paper examines whether it is possible to identify a driver from a small number of OBD2 sensors data collected from 8 persons driving one vehicle under different conditions. For this purpose WEKA program and its available models were used. Collected parameters were rated using InfoGainAttributeEval attribute evaluation model. Using the 6 highest rated attributes and the Random Forest algorithm it is possible to classify a driver with a 90% probability from collected OBD2 data. Aditionaly higher frequency data is collected to show, that it is possible to identify number of drivers from driving data collected.

Keywords: data mining, OBD2, drivers identification, u-shapelet, time-series.

AUTOREFERĀTS

Pirmajā darba daļā tiek replicēta pieeja, kas tiek balstīta uz trīs līdzīgām pieejām no zinātniskiem rakstiem. Lai to paveiktu, ir atrasti dati, kas būtu derīgi šim nolūkam. Dati tiek analizēti un apstrādāti, lai spētu pielietot tos izpētes veikšanai. Izpētē netiek lietota konkrēta pieeja no viena darba, bet tiek izmantotas idejas no vairākiem darbiem, lai panāktu labāku rezultātu.

Otrajai darba daļai dati tiek vākti paša spēkiem reālajā pasaulē. Datu vākšanai bija nepieciešamas speciālas iekārtas un programmatūras instaļēšana, kā arī pareizu parametru atrašana, kas aizņem papildus laiku. Datu vākšana bija diezgan laikietilpīga, jo bija jāatrod arī brīvprātīgie, kas bija gatavi piedalīties datu vākšanas procesā. Protams tam sekoja datu apstrāde un tīrīšana.

Otrajā darba daļā tiek praktiski implementēts un pilnveidots algoritms, lai izmēģinātu jaunu pieeju datu analīzei. Algoritma izstrādē tiek apvienotas idejas no vairākām zinātniskām publikācijām. Praktiskā implementēšana un atkļūdošana arī sanāk laikietilpīgs process. Galu rezultātā ir izdevies izveidot algoritmu, kas papildina šobrīd esošās metodes apskatāmās problēmas risināšanai.

Literatūras avotos ir izmantoti 10 zinātniskie raksti, kas sastāda aptuveni pusi no visas izmantotās literatūras.

SATURS

ANOTĀCIJA....................2
ABSTRACT....................3
AUTOREFERĀTS....................4
SATURS....................5
APZĪMĒJUMU SARAKSTS....................6
IEVADS....................7
1. SITUĀCIJAS APSKATS....................9
1.1. Tehnoloģiju apskats....................9
1.2. Veikto pētījumu apskats....................12
2. METODOLOĢIJA....................15
2.1. Uzraudzīta mācīšanās....................15
2.2. Neuzraudzītā mašīnmācīšanās....................16
2.2.1 K-vidējo....................17
2.2.2 “Elkoņa” metode....................20
2.2.3 U-shapelet....................22
3. DATI....................26
3.1 Dati uzraudzītai metodei....................26
3.1 Dati neuzraudzītām metodēm. 29
4. REZULTĀTI UN DISKUSIJA....................31
4.1 Uzraudzītās metodes....................31
4.2 Neuzraudzītās metodes....................34
4.2.1 K-vidējo....................34
4.2.2. U-shapelet....................36
SECINĀJUMI....................38
IZMANTOTĀ LITERATŪRA UN AVOTI....................39
PIELIKUMI....................41
Pielikums nr. 1....................41
Pielikums nr. 2....................42
Pielikums nr. 3....................43
Pielikums nr. 4....................44
Pielikums nr. 5....................45
Pielikums nr. 6....................47
Pielikums nr. 7....................48
Pielikums nr. 8....................50
Pielikums nr. 9....................53
Pielikums nr. 10....................60
Pielikums nr. 11....................64

APZĪMĒJUMU SARAKSTS

CAN - (no angļ. val. Controller Area Network - kontroļu apgabala tīkls) transportlīdzekļu kopņu standarts, kas ļauj mikrokontrolieriem un citām ierīcēm sazināties ar centrālo kontroles datoru.

Decision Tree - (no angļ. val. Decision Tree - lēmuma koks) datu klasifikācijas algoritms.

InfoGainAttributeEval – modelis, kas vērtē atribūtu reitingu balstoties uz to informatīvo ieguldījumu labākai atkarīgā parametra klasifikācijai.

K-MEANS - (no angļ. val. K-MEANS - K – viduspunkti) datu klasifikācijas algoritms, ko izmanto, lai atrastu klases, kurās vislabāk var sagrupēt dotos datus.

KNN - (no angļ. val. K – Nearest Neighbours – K- tuvākie kaimiņi) datu klasifikācijas algoritms.

OBD2 - (no angļ. val. On-board diagnostics. - borta diagnostika) obligāts standarts attīstītajās pasaules valstīs automobiļu tehniskā stāvokļa un izmešu diagnosticēšanai.

Random Forest - (no angļ. val. Random Forest – izlases mežs) datu klasifikācijas algoritms.

U-shapelet – U apzīmē “Unsupervised”, kas no angļ. val ir bez uzraudzības, bez skološanas. Shapelet (netulkojas) neliels apgabals no laika rindas, kas raksturo šo laika rindu un atšķir to no citām. U-shapelet – neliels apgabals no laika rindas, kas to raksturo un ir atrasts izmantojot bezuzraudzības metodes.

Min - max – datu normalizēšanas algoritms, kas balstās uz minimālo un maksimālo vērtību. Normalizē datus intervālā no 0 līdz 1.

python – programmēšanas valoda

IEVADS

Mūsdienu transporta līdzekļi, kas tiek tirgoti Eiropas Savienotajās valstīs, kopš 2003. gada un kopš 1996. gada Amerikas Savienotajās Valstīs ir obligāti aprīkoti ar OBD2 standartu[5], ar kura palīdzību ir iespējams samērā vienkārši vākt datus no dažādiem transportlīdzekļa sensoriem reālajā laikā un glabāt tos apstrādei. Šādus datus mūsdienās vāc apdrošinātāji un autoražotāji savām vajadzībām. Lai gan autoražotāji apgalvo, ka datu vākšana tiek veikta autotransporta stāvokļa monitoringam, un apdrošinātāji neatklāj publiski, kam tieši tiek vākti OBD2 dati, ir aizdomas, ka šie dati var saturēt personiskus datus. Citādi sakot, ir iespējams diferencēt vai identificēt braucējus. Uzticamus modeļus, kas spētu identificēt braucēju no OBD2 datiem, būtu noderīgi gan apdrošinātājiem, ja pārdod personiskās polises, auto iznomātājiem, kā pierādījums ceļa satiksmes negadījumos, soda saņemšanas gadījumos, kā arī pretaizdzīšanas sistēmām, kas spētu identificēt, ka esošais braucējs nav reģistrēts transportlīdzekļa vadītājs.[1]

Pēdējos gados attīstītie datizraces rīki un tehnoloģijas ir vairojušas interesi par dažādiem datiem, tai skaitā arī par OBD2 datiem un iespējamo to pielietošanu braucēja identificēšanā. Publicēti ir parādījušies darbi, kuros tiek pētīta šī problēma, un ir pierādīts, ka, ja dati tiek vākti kontrolētos apstākļos, tas ir, braucot pa vienādiem maršrutiem noteiktajā laikā, lai mazinātu satiksmes ietekmi ir iespējams precīzi identificēt braucēju, ja par to tika iepriekš ievākti dati. [1][3][4]

Šajā darbā ir apskatīti iepriekš veiktie pētījumi un uzstādīts mērķis pārbaudīt vai ir iespējams, izmantojot līdzīgu pieeju, identificēt braucēju, ja dati tiek vākti nekontrolētā eksperimentā, kur automobilis tiek vadīts dažādos maršrutos un apstākļos. Kā arī pētīts vai ir iespējams diferencēt viena automobiļa vadītāju skaitu iepriekš nezinot, cik autovadītāju dati ir savākti.

Izmantojot OBD2 sensoru datus, kas tika ievākti no 8 dažādiem braucējiem vadot vienu transportlīdzekli dažādos apstākļos, ir noteikts, ka pielietojot datizraces algoritmus ir iespējams ar 90% precizitāti identificēt braucēju. Izpētes laikā arī tika atklāts, ka visu sensoru datu iekļaušana nepalīdz labāk identificēt braucēju, bet gan analizējot parametrus un izmantojot tikai tos, kas dod lielāko informatīvo devumu klasificēšanai, ir iespējams panākt labāku modeļu darbību.

Otrā darba daļā tiek apskatīta braucēju identificēšana izmantojot neuzraudzītās

datizraces metodes. Šim nolūkam vajadzēja ievākt jaunus datus ar biežāku ierakstīšanas intervālu. Datu ievākšanas process ir detalizēti prezentēts 3.1. apakšnodaļā. Līdzšinēji darbi, kas veiktu braucēju identificēšanu no OBD datiem izmantojot neuzraudzītās metodes netika atrasti. Šim nolūkam tika izvēlēts u-shapelet algoritms, kas spēj klasificēt laika rindas izmantojot raksturīgus apgabalus laika rindās, to sadalīšanai. [12] Algoritms tiek implementēts python valodā un papildināts ar "elkoņa" metodi klasteru skaita atrašanai. [19] Izveidotais algoritms, analizējot braukšanas ātruma laika rindas spēj precīzi atrast braucēju skaitu ievāktajos datos, ja to skaits ir virs 1. Pētījumā ievāktie dati un python implementācija ir atrodama – https://github.com/vavere00/OBD2_driver_identify.

1. SITUĀCIJAS APSKATS

1.1. Tehnoloģiju apskats

Mūsdienu automobiļos izmantotās tehnoloģijas padara to vairāk nekā tikai par pārvietošanās līdzekli. Viena no šādām tehnoloģijām ir OBD2 standarts. OBD (no angļ. val. On-board diagnostics. - borta diagnostika) ir termins, ko lieto, lai apzīmētu automobiļu pašmonitorēšanas sistēmas, kas ļauj lietotājam, izmantojot specializētu portu, spēt nolasīt kļūdas. Mūsdienās šis standarts ir attīstījies līdz OBD2, kur ir noteikti standartizēti OBD-II PID kodi, kas ir vienādi visiem auto ražotājiem. [11] Kā arī ir noteikti standartizēti OBD2 fiziskā porta izeja. OBD2 standartu 1994. gadā ieviesa CARB Kalifornijas tīra gaisa aģentūra, un no 1996. gada tas ir obligāts visās Amerikas Savienotajās Valstīs (ASV). Eiropas Savienībā (EU) no 2003. gada visiem pārdotajiem automobiļiem ar benzīna vai dīzeļdegvielas iekšdedzes dzinējiem obligāti ir jābūt aprīkotiem ar OBD2. [5] Standarts tika ieviests, lai var ātri mērīt automobīļu gāzu izmešus gaisā standartizētā veidā. Kā arī apskatīt iekārtu kļūdas, kas saistās ar gaisa izmešiem, kā arī mērīt degvielas patēriņu.

OBD2 nav sava atsevišķa tīkla, tas izmanto signālus no transportlīdzekļa CAN bus tīkla. CAN (no angļ. val. Controller Area Network - kontroļu apgabala tīkls) transportlīdzekļu kopņu standarts, kas ļauj mikrokontrolieriem un citām ierīcēm sazināties ar centrālo kontroles datoru. [2] Lai gan CAN signāli ir tie, kas būtībā komunicē ar ierīcēm un dod plašāku iespēju klāstu, tie nav standartizēti un atšķiras katram ražotājam. Līdz ar to darbs ar tiem ir daudz grūtāks nekā ar OBD2, kur viss ir standartizēts. Tāpēc šajā darbā tiks apskatīti tikai tie dati, ko ir iespējams savākt izmantojot OBD2 no jebkura EU pārdota automobiļa pēc 2004. gada. Eiropas savienībā OBD2 standarts, Eiropā dēvēts par EODB (European OnBoard Diagnosis – no angļ. val. Eiropas Borta Diagnostika). Obligāts visiem Eiropas Savienībā pārdotajiem automobiļiem ar 2000. gadu aprīkotiem ar benzīna iekšdedzes dzinējiem un no 2003. gada arī dīzeļdegvielas darbināmiem transporta līdzekļiem. Tā kā OBD2 portam ir izejas arī uz CAN tīklu, mūsdienās, daudzos automobiļos ir iespēja nolasīt, ne tikai standarta OBD2 signālus, bet arī citus mērījumus no sensoriem, kas nav tieši saistīti ar izmešu daudzumu. Piemēram mašīnām ar elektrisko stūres un pedāļu kontroli, var nolasīt arī šādus mērījumus, no OBD2 porta, bez speciālas iejaukšanās CAN mašīnas tīklā. Šajā darbā uzmanība tiek pievērsta tieši OBD2 standartam, jo augšup minētie speciālie parametri nav pieejami visiem transportlīdzekļiem, savukārt OBD2 ir likumā noteikts standarts, līdz ar to iegūtie rezultāti var būt plašāk pielietoti.

Mūsdienās ir daudz dažādu veidu, kā vākt informāciju no OBD2. Sākot no informācijas vākšanas blokiem, kas tikai vāc signālus braukšanas laikā un ieraksta to datu nesēja, līdz beidzot ar pieslēgšanos ar Bluetooth portu un sasaisti ar viedtālruni. OBD2 vākšanu šobrīd jau veic auto apdrošināšanas kompānijas, piedāvājot atlaidi, par tāda pakalpojuma izmantošanu. [3] OBD2 datus mēdz arī vākt dažādas nozīmes komerciālie autoparki, lai sekotu līdzi savu darbinieku braukšanas veidam (interesē ekonomiska braukšana un drošība, kā arī ceļu satiksmes noteikumu ievērošana). Dati tiek vākti un raidīti uz centrāliem apstrādes un glabāšanas serveriem. Ir iespēja arī izmantot auto ražotāju piedāvātās datu apstrādes un viedtālruņu piedāvātās speciālās iespējas. [3]

Lielais datu apjoms, ko ir iespējams savākt standartizētā veidā atver plašas iespējas, izmantojot mūsdienu datu apstrādes tehnoloģijas, pētīt sakarības un veidot jaunus risinājumus, kas spētu atvieglot vai bagātināt automobiļa vadīšanas pieredzi. Piemēram, tiek piedāvāts risinājums, apvienojot OBD2 datus par gaisa spilvenu nostrādāšanu un pārslodzes datus no viedtālruņa, kur tiek uzstādīta speciāli izstrādāta aplikācija, ir iespējams noteikt īpaši smagus vadāmā automobiļa negadījumus. Šādos negadījumos ir liela iespējamība, ka autovadītājs un pasažieri zaudē samaņu vai ir iesprostoti un nav spējīgi aizsniegties līdz viedtālrunim, lai izsauktu nepieciešamos dienestus. Izveidotais risinājums spēj 3 sekunžu laikā noteikt negadījuma esamību un automātiski par to ziņot attiecīgajām iestādēm, paziņojot arī atrašanās vietu. [7] Šādi risinājumi spētu palīdzēt cilvēkiem, kas iekļūst stiprā satiksmes negadījumā uz apvidus ceļiem ar mazu satiksmi vai vēlajās nakts stundās, kad nav citi satiksmes dalībnieki, kas spēj noreaģēt un izsaukt attiecīgos dienestus.

Šajā darbā tiks apskatīts OBD2 datu izmantošanu ar iespēju identificēt braucēju, izmantojot kādu no mašīnmācīšanās metodēm. Iespējamie izmantošanas virzieni ir darba "flotes" kontrole, piemēram taksometru parkā vai tālbraucēju šoferiem, lai netiek izmantotas citu darbinieku kartes un pārkāpti atpūtas stundu standarti. Apdrošinātāji mēdz tirgot apdrošināšanas polises, uz konkrētu vadītāju un ir ieinteresēti zināt, kas atradās pie stūres nelaimes notikšanas brīdī. Īrētām mašīnām bieži vien nosaka, cik autovadītāji drīkst vadīt automobili, kas arī mēdz būt saistīti ar apdrošināšanas noteikumiem. Tādos gadījumos būtu interesanti zināt cik cilvēki patiesībā ir izmantojuši automobili, lai iegūtu patiesākus datus un iespējams izvērtēt savu piedāvājumu nosacījumus. Piemēram vienmēr ļaut braukt vairākiem vadītājiem, jo tas tik un tā tiek darīts. Līdzīgus datus būtu interesanti zināt arī kompānijām un valsts iestādēm, kas ļauj izmantot dienesta auto. Kā arī ir iespējams veidot dažādus pret aizdzīšanas risinājumus, kur no OBD2 datiem tiek noteikts, ka vadītājs nav reģistrēts auto vadītājs, pēc kā tiek bloķēts dzinējs, sūta paziņojumu uz viedtālruni. [1]

Ir izstrādāti vairāki darbi[1][3][4], kuros autori ir veiksmīgi izdevies atrast modeļus, kas ar vairāk kā 98% precizitāti spēj atpazīt braucēju, sagatavotajos datos. Darbos ir atrastas labas pieejas un strādājoši modeļi, taču tie ir iegūti uz “ideālos” apstākļos iegūtiem datiem. Šajā darbā ir mērķis apzināt idejas no iepriekš veiktajiem pētījumiem un paskatīties uz to efektivitāti pielietojot “reālākiem” datiem.

Nākamajā daļā tiks apskatīti daži darbi, kur autori ir strādājuši pie autovadītāja noteikšanas izmantojot datizraces paņēmienus un OBD2 datus.

1.2. Veikto pētījumu apskats

Pirmais darbs, kas tiek apskatīts ir Miro Enev u.c. Automobile Driver Fingerprinting", kas savā darbā pēta, cik daudz ir iespējams pateikt, par autovadītāju izmantojot tikai datus, ko iespējams vākt ar OBD2. Autoru bažas ir, ka dati, ko vāc apdrošināšanas kompānijas, mašīnu ražotāji un viedtālruņu piedāvātās aplikācijas, ir personalizējami un pastāv iespēja no tiem interpretēt personiskos datus. [3]

Lai savāktu datus autori veica eksperimentu, kurā piedalījās 15 braucēji, no kuriem 7 vīrieši un 8 sievietes. Braucēji bija iepriekš iepazinušies ar transportlīdzekli un braukšanas laikā klausījās vienu radiostaciju. 2 braukšanas posmi: a) manevrēšana pa slēgtu stāvvietu, b) 80 km noteikts maršruts pa Sietlu vienādā diennakts laikā. OBD2 dati no 16 sensoriem. No tiem tika izmantoti 15, aprakstu skatīt pielikumā Nr. 1. Lai noteiktu efektīvāko parametru, katrs parametrs tika testēts viens pats uz visiem datiem izmantojot Random Forest metodi. Nozīmīgākais parametrs izrādījās bremžu pedālis. Pārējo parametru reitingu skatīt pielikumā Nr. 1. Vāktie dati tika grupēti nelielos sekunžu intervālos un intervāla vidējais izmantots, kā parametra vērtība. Izmantotie datizraces modeļi: Support Vector Machine, Random Forest, Naive Bayes, k-nearest neighbor. Trenēšana-testēšana notika uz visiem datiem izmantojot 90/10 savstarpēju apstiprināšanu (cross validation). Autoru darbā labāk sevi parādīja Random Forest klasifikators un rezultāti, ko viņi ieguva bija braucēja noteikšana: a) tikai pēc bremžu pedāļa pozīcijām – 87.33% precizitāte pēc 15 min braukšanas uz šosejas un 100% pēc 1.5 h uz šosejas, b) 91.33% precizitāte braucēja noteikšanā izmantojot visus parametrus 8 minūšu manevrēšanā pa stāvvietu, c) 100% precizitāte izmantojot visus datus pēc 1.5 h braukšanas pa šoseju un pilsētu. [3]

Byung Il Kwak u.c. savā darbā piedāvā risinājumu auto zādzību gadījumā, ar iespēju identificēt braucēju izmantojot OBD2 automobiļa datus un signāla sūtīšanu uz serveri, ja braucējs netika identificēts, kā saimnieks. [1] Tiek izmantota līdzīga pieeja, kā Miro Enev darbā "Automobile Driver Fingerprinting". 10 braucēji, kas vada KIA markas transportlīdzekli pa 4 dažādie maršrutiem Seulā, kas sevī ietver braukšanu pa pilsētu, šoseju un stāvvietu. Maršrutu kopējais garums – 23 km. Braukšanas eksperiments tiek veikts vienmēr laikā no 20:00 līdz 23:00, lai kontrolētu satiksmes ietekmi. [1] No ievāktajiem datiem pilnajā modelī arī tiek analizēti 15 rādītāji no 51 pieejamiem. Otrajā etapā izmantojot datizraces programmatūru WEKA atribūtu vērtēšanu InfoGainAttributeEval ar Ranker

meklēšanas metodi, tiek noteikti ietekmīgākie parametri, un modelis tiek pārrēķināts. Ar 15 parametriem var iepazīties pielikumā nr. 2., un svarīgākie pēc analīzes izrādījās ilgtermiņa degvielas korekcijas banka un transmisijas eļļas temperatūra. Klāt visiem 15 parametriem tiek rēķināts to statistiskās vērtības (vidējā vērtība, mediāna, standarta novirze) intervāla 60 sekundes. Tādā veidā klāt 15 parametriem rodas vēl 45 parametri, kopā 60. Klasifikācijai izmanto Decision Tree, Random Forest, KNN, un MLP datizraces algoritmus. Modeļu trenēšanai un testēšanai izmanto visus datus 90/10 sadalījumā ar 10-kārtējo starpvalidāciju (10-fold cross-validation). Modeļu rezultāti apskatāmi tabulā nr. 1.1. Secinājums ir, ka ar lielu varbūtību ir iespēja noteikt braucēja identitāti, taču, ar 60 sekunžu intervālu pieeju izmantošanu, lai iegūtu statistiskos parametrus, kas būtiski paaugstina modeļa precizitāti, ir nepieciešamas vismaz 60 sekundes, lai iegūtu pirmos rezultātus.

1.1 tabula

Modeļu rezultāti

Ceļa tips	Decision Tree	KNN	Random Forest	Multilayer perceptron
Pilsēta	0.987	0.963	0.998	0.948
Šoseja	0.990	0.984	0.998	0.989
Stāvvieta	0.978	0.925	0.993	0.956

Fabio Martinelli u.c savā 2018.gada darbā "Human Behavior Characterization for Driving Style Recognition in Vehicle System" analizēja tos pašus OBD2 datus no Seulas Dienvidkorejā ar mērķi noteikt autovadītāju, izmantojot datizraces modeļus. [4] Pieeja ir sākumā testēt modeļus ar visiem 51 parametriem, pēc tam izmantojot BestFirst un GreedyStepwise atlasīt 6 nozīmīgākos un izmantot tikai tos. Izmantotie datizraces algoritmi – J48, J48graft, J48consolidated, RandomTree, RepTree. Visātrāk strādāja un visprecīzākos rādītājus ap 99%, gan ar pilno parametru kopu, gan ar 6 nozīmīgākajiem parādīja J48 algoritms. Seši nozīmīgākie parametri: Ieplūdes gaisa spiediens, dzinēja stāvēšanas laiks, ilgtermiņa degvielas padeves korekcija, berzes griezes moments, transmisijas eļļas temperatūra, stūres ātrums. [4] Tabula angļu valodā pielikumā nr. 3.

No apskatītiem darbiem var redzēt, ka kontrolētā eksperimentā ir iespējams noteikt braucēju ar diezgan lielu precizitāti, izmantojot tikai OBD2 datus un datizraces metodes. Kā arī nav nepieciešami ļoti daudz parametri, galvenais, lai ir pieejami noteicošie. Kas nozīmē, ka laba stratēģija ir mēģināt noteikt šos galvenos parametrus, lai atvieglotu modeli un skaitļošanas laiku. Kā arī tas varētu palīdzēt izvairīties no pārākpielāgotības. Efektīvākie modeļi sevi parādīja J48, Random Forest, Random Tree un k-nearest neighbours.

Iepriekš apskatītie darbi ir balstīti uz tā saucamām "supervised learning", jeb latviski

uzraudzīta mācīšanās. Kas nozīmē, ka mums sākotnēji ir informācija par pētāmo objektu un tā uzvedību. Šīs zināšanas tiek izmantotas, lai trenētu modeli un nākotnē spētu identificēt iepriekš pētāmos objektus. Savukārt šī darba otrs pētāmais jautājums, vai ir iespējams noteikt autovadītājus zinot tikai ievākto informāciju par veiktajiem braucieniem no automobiļa izmantojot OBD2 standartu. Šādam nolūkam izmanto tā saucamās "unsupervised learning", jeb latviski neuzraudzītās mācīšanās metodes. Tādus darbus, kas mēģinātu klasificēt OBD2 datus ar neuzraudzītām metodēm, atrast nav izdevies.

Tā kā savāktie dati savā ziņā ir laika rindas no katra sensora, šī problēma varētu būt pētīta ar laika rindu klasificēšanas metodēm. Interesantu metodi laika rindu klasificēšanā savā darbā piedāvā Zakaria et.al., kur viņi arī parāda, ka tā ir daudzos piemēros efektīvāka par "klasiskajā" metodēm, tādām kā iezīmju izgūšana (angļu - feature extraction) un Eiklīda distance. [12] Iezīmju iegūšanas metode paredz laika rindas klasificēt pēc to aprēķinātām statistiskām iezīmēm, piemēram, vidējā vērtība, standart novirze, asimetrija utt. Braucienu gadījumā tas var nebūt īpaši efektīva metode, jo atšķirībā no iežu spektrālās analīzes, braukšanas procesā vidējā vērtība un novirze parametram var būt vairāk atkarīga no maršruta un satiksmes, nekā no vadīšanas stila. Eiklīda distance mēra laika rindu tuvību vienu otrai un tādā veidā var sagrupēt. Galvenā problēma ar šo mērījumu ir izvēlēties pareizi vienādus nogriežņus no laika rindas, kurus būtu vērtīgi salīdzināt. Autoru piedāvātā metode ir algoritms , ar kura palīdzību spēt atrast tādu apgabalu laika rindā, kas būtu sastopams vēl kāda citā, kas šīs laika rindas spētu vislabāk atdalīt no pārējās masas. Salīdzinājumu ar iepriekš minētājām metodēm uz dažādām laika rindu datu kopām var apskatīt pielikumā nr. 11. Algoritms nav pilnīgs šajā darbā noteiktajai problēmai, jo tas paredz zināt klašu skaitu.

Nākamajā darba daļā tiks apskatīta metodoloģija uzraudzītām un neuzraudzītām mašīnmācīšanās metodēm, ar kuru palīdzību tiks pētīti dati, kas tika vākti dažādiem braucējiem braucot reālistiskos apstākļos, dažādos ceļos, diennakts laikos un laikapstākļos.

2. METODOLOĢIJA

2.1. Uzraudzīta mācīšanās

Iepriekšējā nodaļā tika apskatīti esošie darbi un izmantotās metodoloģijas OBD2 datu izpētei, lai identificētu braucēju. Atšķirībā no iepriekš minētajiem darbiem, dati, kas tiks izmantoti šajā darbā, nav vākti vienādos apstākļos, tāpēc rezultāts varētu atšķirties. Detalizētāk par datiem un to apstrādi nākamajā nodaļā.

Apkopojot iepriekšējo darbu labās prakses, ir izveidota šī darba metodoloģija. Pēc datu apstrādes tiks pielietoti datizraces algoritmi, kas sevi labi parādīja darbā ar līdzīgu problēmu. Trenēšanai, testēšanai tiks izmantoti visi dati 90/10 - treniņa/testa sadalījums, izmantojot 10-kārtējo validāciju. Modeļu efektivitāte tiek mērīta ar pareizi klasificētu klašu proporciju procentos. Datizraces nolūkiem tiks izmantota programmatūra WEKA[6].

Izvēlēti tika 3 modeļi, kas parādīja labus rezultātus un bija izmantoti vismaz 2 no apskatītajiem darbiem. Tādā veidā ir izmantoti šādi modeļi:

Lēmumu koki (Decision Tree) – konkrētāk J48 implementācija WEKA. J48 parādīja sevi, kā labāko F. Martinelli (2018) darbā [4], un lēmumu koks bija otrs precīzākais algoritms Byung Il Kwak darbā [1].

Izlases mežs (RandomForest) – RandomForest implementācija WEKA. Parādīja precīzāko klasifikāciju divos darbos.[1][3]

K-tuvākie kaimiņi (KNN) - IBk implementācija WEKA. Tika izmantots 2 darbos. [1][3] Neparādīja augstāko rezultātu, taču ir jēga iekļaut arī šo algoritmu, lai tiktu pārstāvētas dažādāki modeļi, ne tikai lēmumu koki.

Iepriekš veiktie darbi[4][3], parādīja, ka ir iespējams, nezaudējot lielu precizitāti, izmantot tikai nozīmīgus parametrus klasificēšanai, kas samazina izpildes laiku un modeļa sarežģītību. Šim nolūkam tiks izmantota WEKA piedāvātais InfoGainAttributeEval atribūtu vērtēšanas modelis. Modelis savus lēmumus balsta uz to, cik vairāk informācijas iegūst par klasi, ja ir zināma pētāmā atribūta vērtību. InformācijasIeguvums(Klase, Atribūts) = H(Klase) - H(Klase | Atribūts), kur H ir entropijas funkcija. [9]

Pēc atribūtu reitinga noteikšanas, modeļus darbina vēlreiz, ņemot ārā atribūtus, sākot ar zemāk novērtēto. Izpētīt vai ir iesējams samazināt atribūtu skaitu stipri neietekmējot modeļa precizitāti.

2.2. Neuzraudzītā mašīnmācīšanās

Otrā darba daļā tiek apskatīta iespēja identificēt braucējus, konkrētāk spēt identificēt braucēju skaitu zinot tikai ievāktos datus, bet nezinot nekādu informāciju par braucējiem. Šāda tipa problēmas tiek definētas kā neuzraudzītās mašīnmācīšanās problēmas, jo modelim nav trenēšanas kopas, uz kā pamata trenēt modeli. Šajā darbā definētajā problēmā ir nepieciešams sadalīt visus braukšanas ierakstus pa klasteriem, kur katrs klasteris būs attiecināms uz vienu braucēju.

Klastera definīcija: Ja dota n objektu kopa D, objektiem ir p atribūti, skaitliski un/vai nomināli, tad klasteris ir kopas D apakškopa, kurā esošie objekti attāluma ziņā ir tuvāk viens otram, salīdzinot ar objektiem ārpus klastera. [13] Kur attālumu nosaka ar kādu no distances mēriem, piemēram Eiklīda distanci daudzdimensionālā telpā. Formula 2.1 parāda kā var aprēķināt Eiklīda distanci starp punktu p un q , n dimensiju telpā.

$$\begin{aligned} d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) &= \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}. \end{aligned} \tag{2.1}$$

d – distance

p – punkts “p”

q – punkts “q”

n – dimensiju skaits

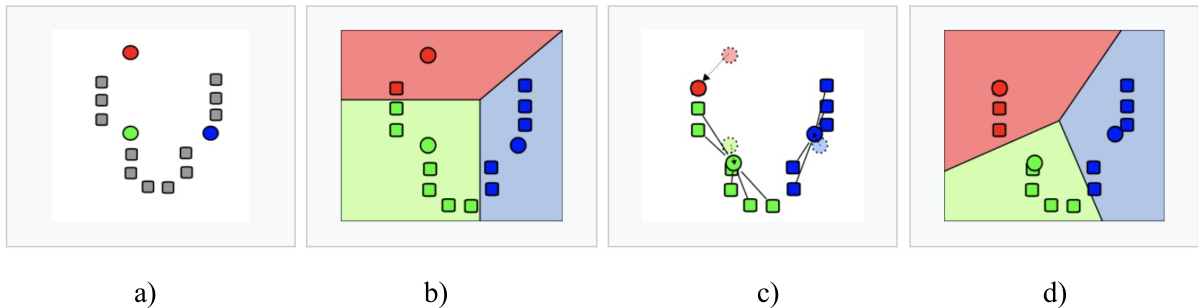
Šajā darbā Eiklīda distances aprēķināšanai, kur tas ir nepieciešams, tiek izmantota python sklearn bibliotēkas funkcija euclidean_distances.[14]

Viens no senākiem , kura idejas sākas 1956. gadā, un plaši pielietojiem algoritmiem, kas tiek izmantots laika rindu klasificēšanai, ir k-vidējo algoritms. [15][16] Tā kā pats algoritms nespēj noteikt labāko klasteru skaitu, tikai to labāko sadalījumu. Klasteru skaitu noteikšanai tiks izmantota tā dēvētā “ceļgala” metode, kas, balsoties uz kopējo distanču summu līdz noteikto klasteru centriem, nosaka optimālāko klasteru skaitu, dotajai datu kopai.

Tā kā braukšanas dati var tikt apskatīti, kā laika rindas no katra atribūta, tad kā otra metode tika izvēlēta tāda metode, kas specializējas tieši uz laika rindu klasificēšanu. Savu metodi Zakaria ir nosaucis par U-shapelet, tas ir neliels laika rindas apgabals, kas ir unikāls laika rindai un laika rindām, kas varētu ietilpt vienā klasē, kā arī šis apgabals tiek atrasts ar neuzraudzītu metodi. [12]

2.2.1 K-vidējo

Neliels ieskats k-vidējo algoritma darbībā un tā pielietošanu šajā pētījumā. K-vidējo algoritms spēj atrast labāko sadalījumu punktu mākonim daudzdimensionāla telpā, ja noteiktam klasteru skaitam. Šajā darbā tiek izmantota K-vidējo implementācija ar python sklearn KMeans. [17] Klasterizācija tiek panākts vairākos soļos. Soļi vizuāli attēloti attēlā nr. 2.1. Detalizētāks apraksts seko zemāk.



a) b) c) d)

2.1. att. **K-vidējo algoritma soļi: [15]**

a) inicializācija, b) sadalīšana, c) jauni centri, d) atkārtošana/apstāšanās

a) inicializācija:

Pirmais solis, kurā tiek izvēlēti tik daudz sākuma punkti, cik ir klasteru daudzums. Pastāv dažādas metodes, kurus punktus izvēlēties, kā klasteru centrus pirmajā solī. Šajā darbā izmantotajā algoritmā sākuma inicializācijas metode ir k-means++. Tā cenšas izvēlēties sākuma punktus relatīvi tālu vienu no otra, kas pēc izstrādātāju dokumentācijas apgalvojumiem dod labāku rezultātu nekā patvaļīga punktu izvēle. Ja ir k – klasteru skaits, tad pirmajā solī tiek iedalīti k centri, kas tiek apzīmēti ar m – $m_1^{(1)},\ldots,m_k^{(1)}$.

b) sadalīšana:

Pēc centru izvēles, notiek punktu sadalīšana pa klasteriem. Tas tiek darīts rēķinot Eiklīda distanci no katra punkta līdz visiem klasteru centriem. Punkts tiek piesaistīts attiecīgi tam klasterim, līdz kura centram distance ir mazāka. Formula 2.2 parāda nedaudz formālāku idejas reprezentāciju.

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \ \forall j, 1 \leq j \leq k\}, \quad (2.2)$$

$S_i^{(t)}$ – klasteris i pēc iterācijas t

p – punktu skaits , kas jāklasificē

x_p – katrs punkts no datu kopas

k – klasteru skaits

m_i – i klastera centrs

m_j – visi pārējie klasteru centri, kas nav i

t – iterācija

c) jauni centri:

Populārākais paņēmiens jaunu centru izvēlei ir vienkārši centra punkta atrašana jaunizveidotajos klasteros un pieņem tos, kā centru nākamajā interācijā t + 1. Formālāk attēlots formulā 2.3.

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2.3}$$

$m_i^{(t+1)}$ – jaunais centrs i-tajā klasterī nākamajā iterācijā t+1

$S_i^{(t)}$ – punktu skaits klasterī i

x_j – visi punkti no klastera i

d) atkārtošana/apstāšanās:

Šajā solī tiek atkārtoti soļi b) un c) līdz jaunie centri periodā t+1 ir vienādi ar centriem periodā t. Citiem vārdiem, modelis stabilizējas. Taču bieži tiek izmantoti papildus stāšanās mehānismi, lai modelis netaisa pārāk daudz iterācijas vai neieiet mūžīgā ciklā. Šajā darbā izmantotajā k-vidējo implementācijā tiek izmantoti 2 papildus apstāšanas mehānismi:

1) maksimālo iterāciju skaits = 300. Tas ir algoritms apstāsies pie esošās klasteru izvēles, ja t = 300.

2) tolerances līmenis = 0.0001. Tas nozīmē, ka ja distanču summas starpība visiem punktiem starp iterāciju t un t-1 ir mazāka, par toleranci (0.0001), tad process apstājas. Maza starpība parāda, starp periodiem parāda, ka algoritms vairs nedod lielu uzlabojumu no sadalījumiem.

Tā kā k-vidējo algoritma rezultāts ir atkarīgs no sākotnējo punktu izvēles. Algoritms tiek laists 10 reizes un izvēlas labāko rezultātu pēc kopējo distanču summas. Papildus tam, ka 10 reižu atkārtošana arī ir implementēta KMeans sklearn klasterizācijas algoritmā.

Iepriekš aprakstīts process atrod labāko sadalījumu noteiktam klasteru skaitam, taču neizvēlas labāko klasteru skaitu. Tādam nolūkam klasterēšanas algoritms tiek laists vairākas reizes ar dažādu klasteru skaitu, no 1 līdz ievākto braucienu skaitam. No uz katru klasteru skaitu tiek darbināts k-vidējo algoritms un izvēlēties labākais (mazākais) distanču summas variants. Savākto distanču summas tiek attēlotas pret klasteru skaitu. Un meklēts punkts kur relatīvā distanču summas izmaiņa no k uz k+1 klasteri ir vislielākā. Aprēķināšana notiek izmantojot python kneed bibliotēku, kas ir balstīta uz Satopaa et.al. (2007) darbu “Finding a

“Kneedle” in a Haystack: Detecting Knee Points in System Behavior”. [18][19] K-vidējo python koda implementācija atrodama pielikumā nr. 8.

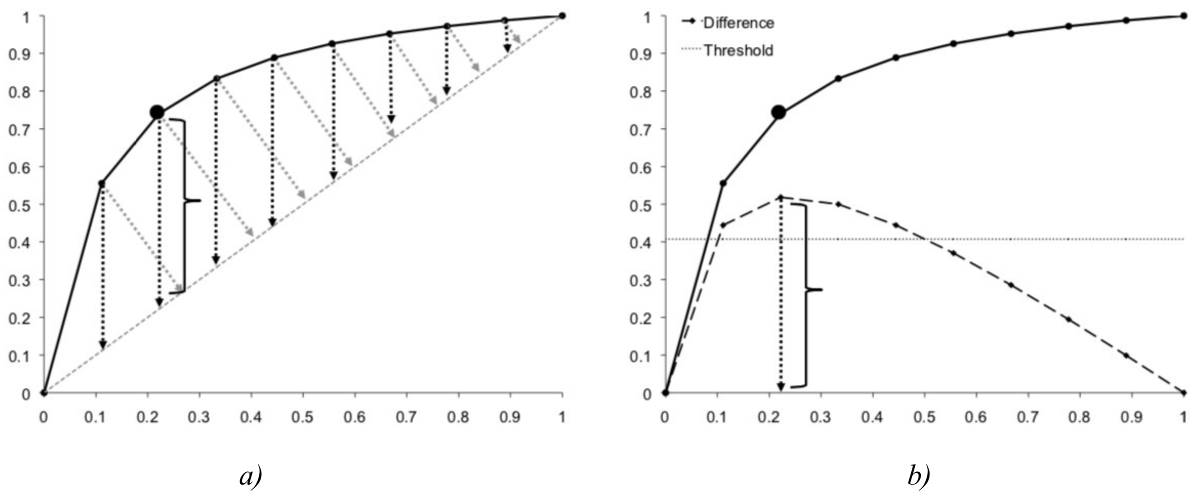
Lai uzlabotu sadalīšanas rezultātus. Tiek pielietotas dažādas datu apstrādes un paplašināšanas metodes, kas ir sevi pierādījušas. Piemēram, ir pierādīts, ka izmantojot vidējo vērtību pa nelielu laika intervālu mēdz uzlabot prognozēšanas modeļu rezultātu uz OBD2 datiem.[1] Konkrēts laika intervāls nav zināms, tāpēc to nāksies noteikt eksperimentālā veidā. Tā kā datiem ir atkarība laikā, tas ir, vienāds mērījums laikā t var raksturot dažādu uzvedību/procesu, atkarībā no tā stāvokļa laikā t-1 , t-2 utt., tiek arī pārbaudīts vai ieviešot papildus parametrus, kas ir mērījumi iepriekšējā laika intervālā. Vai iepriekšējo stāvokļu iekļaušana uzlabo rezultātu un cik daudz iepriekšējo stāvokļu ir jāiekļauj modelī nav zināms un tiks noteikts eksperimentālā veidā. Konkrēti šādas pieejas piemērus atrast nav izdevies, taču šādas metodes ar laika nobīdes parametriem tiek bieži pielietotas laika rindu analīzē un prognozēšanā, piemēram, izmantojot regresijas modeļus un trenējot neironu tīklus, kur slēptie slāņi modelī ir iepriekšējo periodu dati, kas prognozē nākamo periodu utt., piemēram, “Restricted Boltzmann Machines”. [20] Tā kā k-vidējo algoritms skaita distances daudzdimensionālā telpā, dimensijas vai parametri, kuru absolūtās vērtības ir lielākas, var dot lielāku svaru, tāpēc ir ieteicams datus normalizēt, pirms pielieto K-vidējo algoritmu. Mūsdienās pastāv daudz dažādu normalizācijas metožu, kas atšķiras gan savās metodēs, gan vērtībās. Citas normalizē ap 0, atļaujot negatīvas vērtības, citas strikti starp 0 un 1. Dažādas normalizācijas metodes mēdz būt vairāk vai mazāk piemērotas dažādiem mašīnmācīšanās algoritmiem, tāpēc neviena no tām nav universāla. Populārākā metode datu normalizācijas metode, ko lieto kopā ar k-vidējo algoritmu, un kā pierādīts Shalabi darbā, testējot uz dažādām datu bāzēm efektīvākā ir “min-max” metode. [21] Kas arī tiek pielietota šajā darbā. “Min-max” metode attēlota formulā nr. 2.4. Normalizētas vērtības ir no 0 līdz 1.

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{2.4}$$

z_i – normalizēta vērtība no x_i
x_i – i-tā vērtība no apskatāmajiem datiem
min(x) – globālais minimums no visām x vērtībām
max(x) - globālais maximums no visām x vērtībām

2.2.2 “Elkoņa” metode

Šī metode palīdz noteikt optimālo klasteru skaitu, pēc tam, kad ir veiktas vairākas klasterizācijas ar k-vidējo algoritmu datu kopai, pieņemot dažādu klasteru skaitu. Algoritms ņem klasteru skaitu un distanču summas pie noteiktā klasteru skaita, ko uzģenerē k-vidējo algoritms, un mēģina noteikt optimālo klasteru skaitu. Šajā darbā tiek izmantota python kneed implementācija, kas ir bāzēta uz algoritmu, aprakstītu Satopaa V. darbā “Finding a “Kneedle” in a Haystack: Detecting Knee Points in System Behavior”.[18][19] Attēlā 2.2 attēloti algoritma galvenie pamatprincipi.



2.2. att. **Kneedle algoritma pamatprincips [19]**

Uz vertikālās ass tiek attēlotas “min-max” normalizētas distanču vērtības un uz horizontālās ass “min-max” normalizēts klasteru skaits. a) parāda distanču līkni (augšējā līkne) un distances līdz taisnei, kas savieno sākuma un beigu punktu. b) parāda distanču līkni (augšējā līkne) un distanču starpību līkni, kuras maksimums arī nosaka klasteru skaitu. Threshold – slieksnis.

Galvenie algoritma soļi:

a) Klasteru un distanču dati tiek normalizēti ar “min-max” metodi.

b) Ja grafiks ir dilstošs, tad to apgriež izmantojot formulu 2.5.

$$y_i = y_{max} - y_i \, , \, x_i = x_{max} - x_i \qquad (2.5)$$

y_i = vertikālās ass koordināte (distance)
x_i = horizontālās ass koordināte (klasteru skaits)
x_{max} = maksimālais klasteru skaits
y_{max} = maksimālā distance

c) Rēķina vertikālu distanci no distanču līknes līdz taisnei, kas savieno šīs līknes sākuma un beigu punktu. Šī taisne tiek ieviesta, lai tiktu ņemta vērā virzības tendence (augoša, dilstoša funkcija).

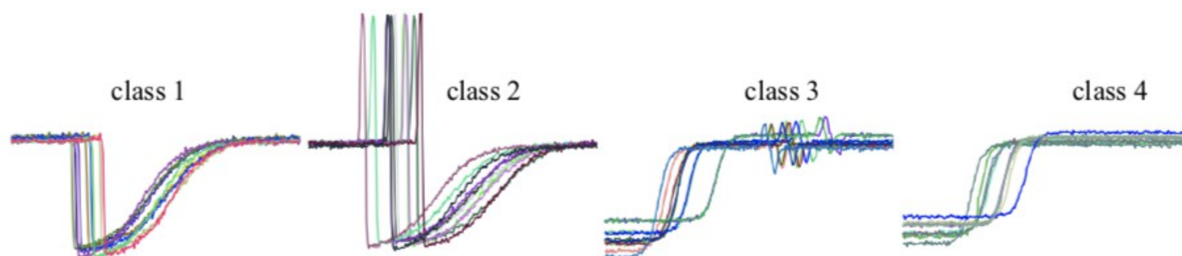
d) Sarēķinātās distances atliek jaunā grafikā un meklē šīs līknes maksimumu pa vertikālo asi. Attiecīgā x koordināte ir optimālais klasteru skaits apskatāmajā datu kopā.

e) Ātrākai darbībai arī tiek izmantots slieksnis. Algoritms apstāsies sasniedzot slieksni otro reizi, un klasteru skaits būs atbilstoši maksimumam līdz sliekšņa sasniegšanai.

Pēc labākā klasteru skaita noteikšanas, attiecīgās klasterizācijas pareizība tiek mērīta ari ARI – Adjusted Rand Index (pielāgots nejaušs indekss), python sklearn.metrics.adjusted_rand_score implementācija. [22] ARI indekss aprēķina līdzības mērījumu starp divām kopām, apsverot visus paraugu un skaitīšanas pāru pārus, kas tiek piešķirti vienā vai dažādās kopās prognozētajos un patiesajos klasteros. Tādējādi tiek nodrošināts koriģētais Rand indeksa lielums, kas ir tuvu 0,0 gadījumam, ja izlases marķējums ir atkarīgs no kopu un paraugu skaita un tieši 1,0, kad kopas ir identiskas (līdz permutācijai).

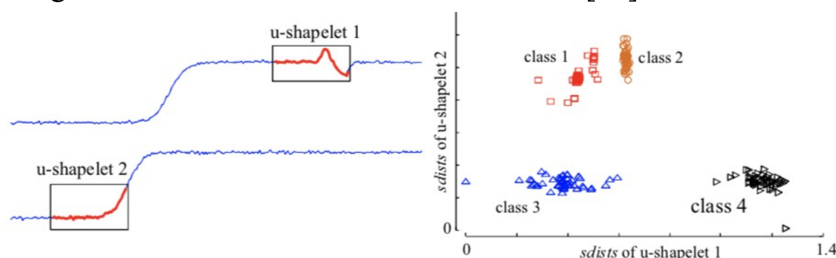
2.2.3 U-shapelet

Kā minēts iepriekš, otrā neuzraudzītā klasterēšanas metode ir laika rindu klasificēšanas metode ir U-shapelet metode. U-shapelet metode meklē nelielas kontūras laika rindā, kas ir raksturīgas dažām laika rindām, un spēj tās atdalīt no pārējo laika rindu datu kopas. Šajā darbā izmantotais algoritms ir balstīts uz Zakaria et.al. 2012. g. darba "Clustering Time Series using Unsupervised-Shapelets". [12] Laika rindu salīdzināšanai visbiežāk izmanto Eiklīda distanci starp divām laika rindām. Problēma ar šo pieeju ir, ka laika rindas ir atšķirīga garuma un ar daudz trokšņiem, kas padara viņu pilno salīdzināšanu diezgan sarežģītu. U-shapelet metode savukārt meklē unikālus apgabalus, kas var būt salīdzinoši neliela garuma pret visu laika rindu. Attēlā nr. 2.3 var novērot, kā vienādas laika rindas no viena klastera ar nelielu nobīdi var būt grūti identificējamas pielietojot Eiklīda distanci pa vertikālo asi.



2.3. att. **Laika rindas ar nobīdi [12]**

Šo problēmu varētu risināt, ja mēs salīdzinātu nevis visu laika rindu, bet tikai nelielu apgabalu no tās, kas to raksturo. Šī ideja attēlota attēlā nr. 2.4. Kur no visas laika rindas tiek izvēlēts apgabals, kas raksturo tieši tās laika rindas, kas ietilpst vienā klasterī. Šie apgabali tiek salīdzināti pret visām laika rindām. Salīdzināšana notiek "bīdot" izvēlēto U-shapelet laika rindu un skaitot Eiklīda distances, minimālā no aprēķinātajām distancēm raksturo salīdzināmās laika rindas tuvību izvēlētajam U-shapeletam. Tātad, katra laika rinda turpmāk tiek raksturota ar minimālo distanci pret katru u-shapelet. Tālāk, jau izmantojot šo distanču karti, ar k-vidējo algoritmu laika rindas tiek dalītas klasteros.[12]



2.4. att. **Klasterēšana ar u-shapelet palīdzību [12]**

Tālāk detalizētāk tiks apskatīts algoritms, kas izmantots šajā darbā. Tas ir balstīts uz Zakaria et.al. 2012. g. darbu, taču ir veikti arī daži labojumi un papildinājumi. Galvenokārt tas saistīts ar to, ka Zakaria et.al. 2012. g. darbā tika pieņemts, ka klasteru skaits ir zināms, bet nav zināms to sadalījums. Savukārt šajā darbā apskatītajā problēmā arī klasteru skaits nav zināms. Otra problēma bija mērījumu skaits, kas ir krietni mazāks, nekā Zakaria et.al. implementācijā. Algoritma radītā python implementācija ir apskatāma pielikumā nr. 9.

Algoritms sākas ar visu laika rindu datu kopu izvēli, minimālo un maksimālo vērtību izvēli nogriežņa garumam, atribūta vārds, kura laika rindas tiks vērtētas.

Sākotnēji pēc patvaļīga principa tiek izvēlēta viena laika rinda. Sākot ar pirmo pozīciju tiek izvēlēts apgabals no laika rindas, kas ir vienāds ar minimāli izvēlēto apgabala garumu. Tālāk šis apgabals, jeb shapelets tiek salīdzināts pret visām laika rindām, bīdot to pa vienai vienībai pa laika rindu un mērot Eiklīda distanci, katrā pozīcijā. Shapelets un laika rindas tiek normalizētas izmantojot z-norm metodi, kas tiek aprēķināta pēc formulas 2.6.

$$z = \frac{x - \mu}{\sigma} \quad (2.6)$$

z – normalizēta vērtība

x – punkts no kopas

μ – punktu vidējā vērtība

δ – punktu standartnovirze

No visām distancēm, kas tiek aprēķinātas, bīdot pa laika rindu, tiek izvēlēta minimālā distance, kas arī tiek piešķirta šai laika rindai, pie šī shapeleta.

Pēc tam, kad tiek izveidots distanču vektors konkrētajam shapeletam, tās tiek sakārtotas augošā secībā. Sakārtotām distancēm katrām divām pēc kārtas ejošām, sākot ar pirmo un nākamo, tiek aprēķināta vidējā vērtība dt. Visas laika rindas tad tiek sadalītas 2 daļās atkarībā vai to distances vērtība ir mazāka vai lielāka par dt. Laika rindas, kuru distances ir mazākas par dt tiek attiecināta uz klasteri A un uz B, ja pretēji. Starp izveidojušām divām grupām tiek aprēķināts attālums starp šīm 2 grupām/klasteriem. Katrā klasteri jābūt vismaz divām laika rindām. Ja ir atdalīta tikai viena laika rinda, tas var drīzāk liecināt par anomāliju, nevis iezīmi. Tādam shapeletam attālumi netiek aprēķināti, un algoritms pāriet uz nākamā garuma shapeleta apstrādi. Attāluma aprēķins notiek pēc formulas 2.7.

$$gap = \mu_B - \sigma_B - (\mu_A + \sigma_A) \quad (2.7)$$

gap = attālums starp klasteriem A un B

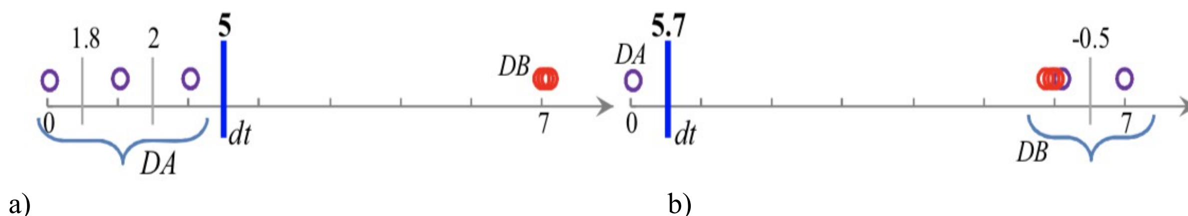
μ_A = klastera A distanču vidējā vērtība

μ_B = klastera B distanču vidējā vērtība

δ_A = klastera A distanču standartnovirze

δ_B = klastera B distanču standartnovirze

Šis attālums tiek saglabāts un process tiek atkārtots visu garuma shapeletiem no minimālā līdz maksimālajam garumam izvēlētajai laika rindai. Kad visi shapeleti ir apskatīti, tad tiek meklēts tāds shapelets, kurš, sadalot laika rindas klasteros A un B pēc distancēm, dod lielāko attālumu starp tiem. Tādā veidā mēs esam ieguvuši klasteri A, kas visvairāk atšķiras no pārējās datu kopas. A un B klašu distances un to sadalīšana ilustrēta attēlā nr. 2.5.



a) b)

2.5. att. **Atdalīšana pēc distancēm [12]**

a) labs sadalījums pēc distancēm apgabalos DA un DB, b) slikts sadalījums, jo redzams, ka DA apgabalā ir tikai viena laika rinda

Tā kā mēs zinām, ka mēs spējam pēc kāda shapeleta atdalīt laika rindas, kas atšķiras no pārējās datu kopas, tad varam tās izslēgt no turpmākās analīzes un meklēt tikai tādus u-shapeletus, kas spēs sadalīt atlikušās laika rindas. Lai netīšām neizslēgtu laika rindas, kas neietilpst klasterī, ko atdala esošais shapelets. Tas ir distance ir mazāka par dt, taču tuvu dt, tāpēc iespējams, ka ir labāki shapeleti, kas izskaidro šo klasi. Tādēļ, lai izslēgtu laika rindas no tālākās apstrādes, tiek izvēlēta konservatīvāka dt vērtība, kas tiek aprēķināta pēc formulas 2.8. Ja laika rindas distance ir mazāka par dt_2 vērtību, tā tiek izslēgta no turpmākiem aprēķiniem.

$$dt_2 = \mu_A + \delta_A \tag{2.8}$$

dt_2 = distances vērtība, lai atdalītu laika rindas

μ_A = klastera A distanču vidējā vērtība

δ_A = klastera A distanču standartnovirze

Tādā veidā tika atrasts pirmais u-shapelets un izņemtas, laika rindas, kuras tas raksturo. Lai izvēlētos nākamo laika rindu, kurai tiks meklēts u-shapelets, atrod tādu laika rindu, kam ir vislielākā distance šim shapeletam. Tas ir tā vismazāk tiek raksturota ar atrasto u-shapeletu, līdz ar to tajā ir vislielākā iespēja atrast nākamo shapeletu. Tā process turpinās, līdz nav iespējas atdalīt vairāk kā vienu laika rindu no pārējā klāsta. Tas ir palikuši trīs vai mazāk laika

rindas.

Dažreiz algoritms atrod triviālus u-shapeletus, kam attālums starp klāsteriem ir 0. Tādos gadījumos netiek saglabāts u-shapelets, rēķināts attālums dt_2 un izņemtas laika rindas. Tādā gadījumā, ar patvaļīgu metodi tiek izvēlēta laika rinda nākamajai apstrādei un process turpinās. Algoritms ir ļoti laikietilpīgs, tāpēc tika ieviesti papildus apstāšanās kritērijs. Tā kā, lai atrastu labu atdalītāju, jāatdala vismaz 2 laika rindas, eksperimentāli pierādījās, ka nav jēgas veikt vairāk kā n/2 iterācijas, kur n ir laika rindu skaits. Lai cīnītos ar problēmu, ka labam sadalījumam ir jāatdala vismaz 2 laika rindas no pārējo klāsta, jo var gadīties, ka ir mums ir jāklasificē no datiem, kas ir, piemērām, tikai gari braucieni, braucieni tiek dalīti īsākās laika rindās, lai izveidotu lielāku datu skaitu klasifikācijai. Tā kā iepriekšējos darbos bija parādīts, ka ir iespējams noteikt braucēju 60 sekunžu laikā ar trenētu modeli, dati tiek dalīti pa 300 sekunžu intervāliem. Ar domu, ka arī šajā gadījumā raksturīgām iezīmēm ir jāizpaužas pa šo laiku. [3]

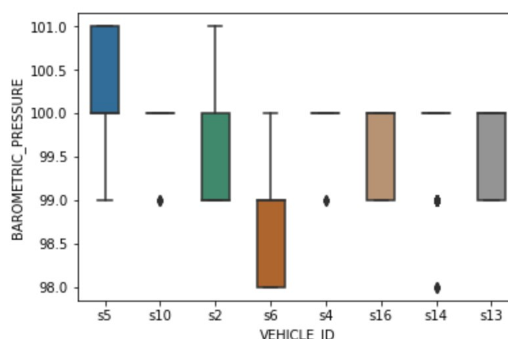
Kad u-shapelet meklēšanas process ir beidzies. Visi veiksmīgie u-shapeleti tiek saglabāti un sākas datu klasificēšanas process. Laika rindas un to distanču vektori, katram u-shapeletam tiek padoti uz k-vidējo klasificēšanas algoritmu, kas pēc šīm vērtībām cenšas klasificēt laika rindas. K-vidējo algoritmu darbina izmantojot klasteru skaitu no 1 līdz par vienu vairāk nekā puse no laika rindu skaita, saglabājot labāko distanču summu no k-vidējās klasifikācijas pie katra klasteru skaita. Tā kā braucēji nevar būt vairāk, kā laika rindu sākotnējos datos, un tās tika sadalītas mazākos intervālos, braucēju nevar būt vairāk kā puse no jaunizveidoto laika rindu skaita. Tālāk, kā jau iepriekš aprakstīts ar "elkoņa" metodes palīdzību tiek atrasts optimālais klasteru skaits. Tādā veidā ir iespējams uzzināt braucēju skaitu un iespējamo braucienu sadalījumu. Sadalījuma kvalitāte tāpat, kā k-vidējo gadījumā tiek mērīta ar ARI indeksu.

Zakaria et.al. 2012. g. darbā uzskata, ka, var atvieglot un uzlabot algoritma darbību, ja izmanto nevis visu u-shapelet distanču matricu klasifikācijai, bet tikai tik daudz, kamēr pievienojot jaunu distanču vektoru, iegūtais klasificējums salīdzinājumā pret iepriekšējo ar ARI indeksu, dod lielāku pieaugumu. Pirmais tiek salīdzināts pret monotonu klasifikāciju. Šajā darbā tiek pielietoti abi paņēmieni un tiks izvērtēts, kurš strādā labāk.

3. DATI

3.1 Dati uzraudzītai metodei.

Dati, kas pētīti šajā metodē ir iegūti no https://www.kaggle.com/cephasax/obdii-ds3#exp2_19drivers_1car_1route.csv . Dati ir vākti ar OBD2 porta palīdzību ierakstot dažādu transportlīdzekļa datus. Dati ir no viena transportlīdzekļa, ko vadīja 19 dažādi autovadītāji dažādos laikapstākļos, dažādu ilgumu, par ceļu nekas nav zināms, tas var būt gan uz šosejas, pagalmos, gan pilsētā ar intensīvu satiksmi. Piemēram, apkārtējās vides temperatūra variē no 18 līdz 35 grādiem pēc Celsija skalas. Sākotnējos datos ir 27 parametri un braucēja identifikators. Daži no parametriem bija doti tukši vai konstanti, no tādiem nācās atteikties uzreiz. Daži rādītāji, piemēram, zemes koordinātes un augstums virs jūras līmeņa, arī tika izslēgti, jo neiekļaujas standarta OBD2 datos. Vēl daži parametri bija tādi, kurus braucējs nevarēja raksturot, bet varēja būt korelēti ar braucēju. Piemēram, degvielas daudzums, atmosfēras spiediens, atmosfēras gaisa temperatūra. Gaisa spiediena un braucēja atkarību var novērot attēlā nr. 3.1.



3.1. att. **Gaisa spiediena un braucēja atkarība.**

Vertikālā ass attēlo gaisa spiedienu kPa, uz horizontālās ass atlikti braucēju identifikatori.

Bija aizdomas, ka apkārtējās vides gaisa temperatūra var ietekmēt arī tādus mērījumus kā uz dzinēju padotā gaisa temperatūru un dzinēja dzesēšanas šķidruma temperatūru, tika veikta korelācijas pārbaude starp šiem parametriem, kas parādīja, ka stipras sakarības nav (tabula nr. 3.1). Līdz ar to var domāt, ka šie rādītāji tiek ietekmēti vairāk ne ar apkārtējās vides apstākļiem.

3.1. tabula

Korelācijas matrica vides temperatūrai.

	AMBIENT_AIR_TEMP	AIR_INTAKE_TEMP	ENGINE_COOLANT_TEMP
AMBIENT_AIR_TEMP	1.000000	0.342325	-0.065378
AIR_INTAKE_TEMP	0.342325	1.000000	0.319304
ENGINE_COOLANT_TEMP	-0.065378	0.319304	1.000000

Ar detalizētu parametru sarakstu, skaidrojumiem un tulkojumiem var iepazīties pielikumā nr. 4. Tā kā sākotnējie datu apzīmējumi bija angļu valodā un apstrādāti tika ar tādiem pašiem parametru nosaukumiem, turpmāk tabulās tie parādīsies ar tādiem pašiem apzīmējumiem. Neskaidrību gadījumā, lūdzu, meklēt skaidrojumus pielikumā nr. 4.

3.2. tabula

Braucēju ierakstu skaits

Braucēja ID	Ierekstu skaits
s12	627
s15	580
s8	521
s11	497
s2	449
s16	446
s13	446
s10	441
s6	436
s5	422
s14	417
s4	416
s7	406
s19	400
s18	360
s1	299
s17	170
s9	142
s3	120

Izvēlētie dati bija ar daudz trūkstošām vērtībām. Vienīgā vērtība, ko izdevās atjaunot ir dzinēja darbības laiks, kur tika reģistrēts laiks ik pēc 4 sekundēm. Dati tika vākti katras 4 sekundes. Līdz ar to trūkstošās vērtības varēja aizvietot ar iepriekšējā ieraksta vērtību + 4 sekundes. Laiks tika konvertēts no laika formāta 'S:M:s' uz sekundēm no sākuma. Pārējās rindas ar trūkstošām vērtībām nācās izdzēst.

Pēc nelietderīgo kolonnu izmešanas, dzinēja laika korekcijas un tukšo vērtību izmešanas, tika apskatīts braucēju ierakstu skaits. Redzams tabulā nr.3.2. Attiecība starp lielāko un mazāko ierakstu skaitu ir ~ 5.2 reizes, kas var negatīvi ietekmēt modeļu trenēšanu.

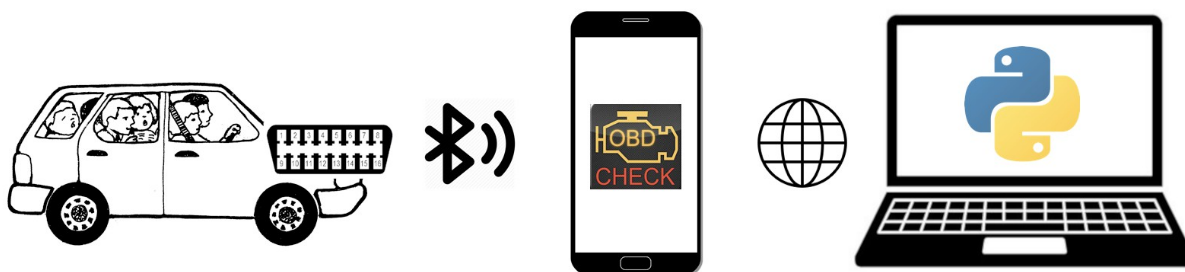
Tā kā trūkstošo vērtību aizstāšana var sabojāt datu struktūru, un no braucēju datiem ar mazu mērījumu skaitu tik un tā nāktos atteikties, tika pieņemts lēmums izvēlēties mazāk braucēju datus, kuru ierakstu skaits tik daudz neatšķiras. Izvēlētie braucēji tabulā nr. 3. ir atzīmēti pelēkā krāsā. Kā redzams, lielāko ierakstu skaits pret mazāko attiecība ir 1.08%. Kas likās pieņemami ņemot vērā, ka pavisam ir 8 identificējamas klases.

Vizuālu reprezentējamu, kā parametru mērījumi izskatās sadalīti pa braucējiem var apskatīt pielikumā nr. 5. Nākamajā daļā tiks apskatīti eksperimentu ar dažādiem datizraces modeļiem rezultāti. Kā arī tiks izpētīts, kā mainās modeļa uzvedība, ja tiek ņemti ārā nenozīmīgi parametri.

3.1 Dati neuzraudzītām metodēm.

Otrajai darba daļai datus vajadzēja vākt paša spēkiem, jo dati, kas izmantoti pētot uzraudzītos modeļus nav pārāk piemēroti laika rindu pētīšanai. Tam ir vairāki iemesli. Pirmkārt, datu ierakstīšanas biežums ir 4 sekundes, kas manuprāt ir par maz, lai uz ceļa spētu ierakstīt unikālus apgabalus, ar kuru palīdzību spēt klasificēt braucējus. Otrs iemesls ir tas, ka datos bija pietiekami daudz pārrāvumu, kas sabojā laika rindas īpašības. Trešais iemesls, kuru varbūt ne īpaši sanāca arī atrisināt, jo dažiem vadītājiem ir tikai pa 2 braucieniem, ir problēma, ka katram braucējam tika ierakstīts tikai viens brauciens, kas var radīt bias estimators, kā piemēram varēja redzēt ar gaisa spiedienu. Veicot vairākus braucienus vienam vadītājam, šādiem parametriem teorētiski būtu jākļūst nenozīmīgiem.

Izmantotais automobilis : Mitsubishi Colt 1.3 l benzīns, 2007.gads. Dati tika vākti izmantojot vienkāršu ierīci, kas no OBD2 mašīnas porta signālu ar Bluetooth palīdzību spēj pārraidīt uz citām ierīcēm (https://www.ebay.co.uk/itm/Bluetooth-Mini-ELM327-OBD2-II-Auto-Car-OBD2-Diagnostic-Interface-Scanner-Tool/264293532804?hash=item3d891f4484:g:j7sAAOSwk7xcvU8x). Šajā gadījumā savienojums notika ar Android viedtālruni. Datu ievākšanai tika izmantota Torque Pro aplikācija (https://play.google.com/store/apps/details?id=org.prowl.torque&hl=en). Tālāk datu apstrāde notiek ar python palīdzību uz datora. Datus nogādāt uz datora var dažādās iespējas, pa vadu, pa e-pastu utt. Ja vēlas sinhronizēt periodiski un automātiski ir iespēja izmantot aplikāciju Syncthing, lai sinhronizētos ar datoru(https://play.google.com/store/apps/details?id=com.nutomic.syncthingandroid), vai Autosync for Google Drive (https://play.google.com/store/apps/details?id=com.ttxapps.drivesync), lai sinhornizētos ar google drive, ja, piemēram, vēlas apstrādāt datus google platformā. Datu vākšanas process shematiski attēlots attēlā nr. 3.2.



3.2. att. **Datu vākšanas process.**

Datu plūsma no kreisās uz labo pusi.

Kopumā, neapstrādātos datos, tika savākti gandrīz 3 h braukšanas datu ar ierakstīšanas frekvenci 1 sekunde. Pietiekami daudz un lietderīgi dati beigās bija 4 dažādiem šoferiem. Dati protams arī ne vienmēr ierakstījās ideāli, bet tad bija iespēja ierakstīt papildus braucienus, bez “caurumiem”, lai rodas nepārtraukta laika rinda.

Datu tīrīšana protams tik un tā bija nepieciešama. Tika izmestas visas konstantās kolonnas, jo tās nedod nekādu papildus informāciju. Tukšās vērtības tika aizstātas ar nulli. Jo pēc loģikas, tas tā arī ir. Piemēram, CO2 izmešu daudzums pie nestrādājoša dzinēja vai ātrums miera stāvoklī. Detalizētāk ar datu apstrādes un rezultāta konvertēšanas uz '.csv' un '.arff' failiem funkcijām var iepazīties pielikumā nr. 10.

Kā minēts iepriekš tika arī aprēķināti jauni atribūti, vidējā slīdošā vērtība pa pēdējām n sekundēm, n izvēle tiks apskatīta eksperimentālajā daļā, kā arī nobīdes vērtības par periodiem t – 1, t – 2 utt. , optimālo nobīdes periodu skaitu un šo parametru efektivitāte tiks apskatīta rezultātos.

4. REZULTĀTI UN DISKUSIJA

4.1 Uzraudzītās metodes.

Ir jau zināms, ka ir iespējams noteikt braucēja identitāti ievāktajiem OBD2 datiem, ja visi braucēji veic vienādu maršrutu.[1][3][4] Šajā eksperimentā ir mēģināts noteikt, cik precīzi var noteikt braucēju, no OBD2 datiem, ievāktiem no viena automobiļa, kur braucēji ir braukuši dažādos laikos, dažādus maršrutus, izmantojot līdzīgas metodes. Konkrētāk ir apskatīts cik labi var prognozēt autovadītāju ar J48, RandomForest un KNN datizraces algoritmiem izmantojot datus ar 12 parametriem, kas aprakstīti pielikumā nr. 4. Algoritmu labākie rezultāti un izpildes laiki prezentēti tabulā nr. 4.1.

4.1. tabula

Rezultāti izmantojot visus 12 parametrus

Metode	Prognozēšanas efektivitāte %
J48	85.63
Random Forest	89.17
KNN	78.54

Kā redzams no rezultātiem, nav jauni pārsteigumi, par modeļu efektivitātes izkārtojumu. Tāpat, kā iepriekš apskatītajos darbos vislabāk sevi parādīja Random Forest, kam seko J48 un KNN. Taču iegūtie rezultāti vidēji ir zemāki. Modeļu konfigurācija programmatūrā WEKA var apskatīt pielikumā nr. 6.

Nākamais solis pētījumā ir atribūtu vērtēšana pēc to informatīvās nozīmes autobraucēju identificēšanā izmantojot WEKA InfoGainAttributeEval atribūtu vērtēšanas algoritmu. Interesanti, ka iegūtie vērtējumi ir atšķirīgi, kā iepriekš veiktajos pētījumos. Piemēram Martinelli u.c (2018) savā darbā, kā nozīmīgāko atribūtu min ieplūdes gaisa spiedienu, kas šajā tabulā ir apzīmēts ar INTAKE_MANIFOLD_PRESSURE un ir novērtēts kā 7. no 12 atribūtiem. [4] Savukārt Miro Enev u.c identificēja ilgtermiņa degvielas korekciju, kas ir agregētā īstermiņa korekcija [4], kā trešo nozīmīgāko parametru [1], bet šeit veiktajos mērījumos īstermiņa korekcija ir pēdējā vietā. Grūti spriest kāpēc tieši tas tā ir, apskatīsim, kā uzvedās modeļi, ja tiek izslēgti mazāk nozīmīgi atribūti. Atribūtu reitings pēc viņu informatīvā nozīmīguma ir apskatāms tabulā nr. 4.2.

4.2. tabula

Atribūtu saraksts pēc to informatīvā nozīmīguma

Informācijas gūšana	Atribūts
1.3289	AIR_INTAKE_TEMP
0.4403	TIMING_ADVANCE
0.3063	ENGINE_COOLANT_TEMP
0.1035	M A F
0.0933	ENGINE_RPM
0.09	ENGINE_RUNTIME
0.064	INTAKE_MANIFOLD_PRESSURE
0.0474	ENGINE_LOAD
0.0435	SPEED
0.0346	THROTTLE_POS
0	SHORT_TERM_FUEL_TRIM_BANK_1

Tabula nr. 4.3. rāda identiski uzstādītus datizraces modeļu rezultātus, izmantojot tikai svarīgākos atribūtus. Kā redzams visos gadījumos ir iespējams panākt modeļu labāku darbību izņemot liekos parametrus. Iepriekš iegūtie J48 un Random Forest modeļi parādīja labāko rezultātu pie 5 svarīgākiem atribūtiem un KNN pie 6. Vienkāršākais izskaidrojums šim varētu būt tas, ka atribūtu samazināšana, samazina modeļu pārākpielāgotību, kas palīdz tiem labāk prognozēt jaunos datus, nevis būt ideāli pielāgotiem treniņu kopai. Modeļu WEKA konfigurācijas atrodamas pielikumā nr. 7.

4.3. tabula

Rezultāti izmantojot labākos parametrus

Metode	Prognozēšanas efektivitāte %	Izmantotie atribūti
J48	86.20	ENGINE_COOLANT_TEMP, ENGINE_RPM, MAF, AIR_INTAKE_TEMP, TIMING_ADVANCE
Random Forest	90.61	ENGINE_COOLANT_TEMP, ENGINE_RPM, MAF, AIR_INTAKE_TEMP, TIMING_ADVANCE
KNN	83.55	ENGINE_COOLANT_TEMP, ENGINE_RPM, MAF, AIR_INTAKE_TEMP, TIMING_ADVANCE, ENGINE_RUNTIME

Iegūtie rezultāti, lai gan nav pārāk slikti, tomēr ir zemāki nekā iepriekš veiktajos pētījumos un, diez vai, ir pietiekami augsti, lai tos varētu izmantot praksē. 1 no 10 gadījumiem tiks klasificēts nepareizi. Arī jāņem vērā, ka veicot 10-kārtējo validāciju dati tiek sajaukti. Lai gan tāda pati metode tika lietota arī iepriekšējos darbos, modelis varētu būt ne tik spēcīgs reālajās situācijā, ja mērķis būtu novērtēt braucēju pēc iespējas ātrāk no braukšanas uzsākšanas brīža. Tādā gadījumā, dati netiek ņemti patvaļīgi no visa brauciena, bet no

braukšanas uzsākšanas un var būt atšķirīgi no vidējā rādītāja.

Iepriekš bija minēts, ka pievienojot papildus statistiskos rādītājus kā papildus parametrus par katra parametra 60 sekunžu logu intervālu, uzlabo modeļa efektivitāti.[1] Šajā darba daļā tas netika apskatīts, bet varētu būt kā nākamais solis.

Pēdējais, ko gribētu piebilst par zemākiem rezultātiem ir tas, ka apstrādātie dati bija ne tikai ievākti ne ideālos apstākļos, bet arī paši dati bija nepilnīgi. Daudzi mērījumi, kā piemēram bremžu pedāļa dati, stūres pagriešanas leņķis un ātrums, vidējais degvielas patēriņš, transmisijas eļļas temperatūra un citi dati, kas bija vērtēti, kā nozīmīgi apskatītajos piemēros nebija ievākti. Turpmākajos darbos būtu vēlams ievākt vairāk OBD2 datus par braucējiem, no kuriem ir iespēja izvēlēties svarīgākos braucēja noteikšanai, tas varētu dot labākus rezultātus. No otras puses šādi dati vairs nevar būt uzskatīti par OBD2 datiem, jo nav obligāti noteikti likumdošanā. Tās ir papildus iespējas, ko piedāvā katrs auto ražotājs, līdz ar to modeļi ar šādiem datiem nevar būt uzskatīti par universāliem reģioniem ar OBD standartu, bet gan konkrētiem auto modeļiem, kam šādi sensori ir pieejami.

4.2 Neuzraudzītās metodes.

4.2.1 K-vidējo

Šajā sadaļā apskatīsimies vai ir iespējams klasificēt datus izmantojot k-vidējo algoritmu, tas ir atrast vadītāju skaitu un noteikt viņu braucienus. Kā arī paskatīsimies vai atribūtu bagātināšana ar vidējo slīdošo vērtību un "atpaliekošās" vērtības dod papildus precizitāti. Sākotnēji izmantosim visus datus no 2 vadītājiem, ar daudz veiktajiem braucieniem, vienam 6, otram 7. Datus normalizēsim ar min-max metodi. Tā kā ir daudz braucieni, ir cerība, ka sanāks izvēlēties tikai tos atribūtus, kas patiešām raksturo braucēju, nevis ir bias novērtējums. Tādus atribūtus, kā brauciena laiks un brauciena distance, uzskatu, ka ir jāizņem pirms atribūtu analīzes, jo tur var arī rasties sakritība, ka tas izskaidro braucēju arī vairāku braucienu gadījumā. Labāko rezultātu pamēģināsim arī uz citu klasteru skaitu. Labākais rezultāts skaitīsies, pareizi noteikts klasteru skaits ar augstāko ARI indeksu.

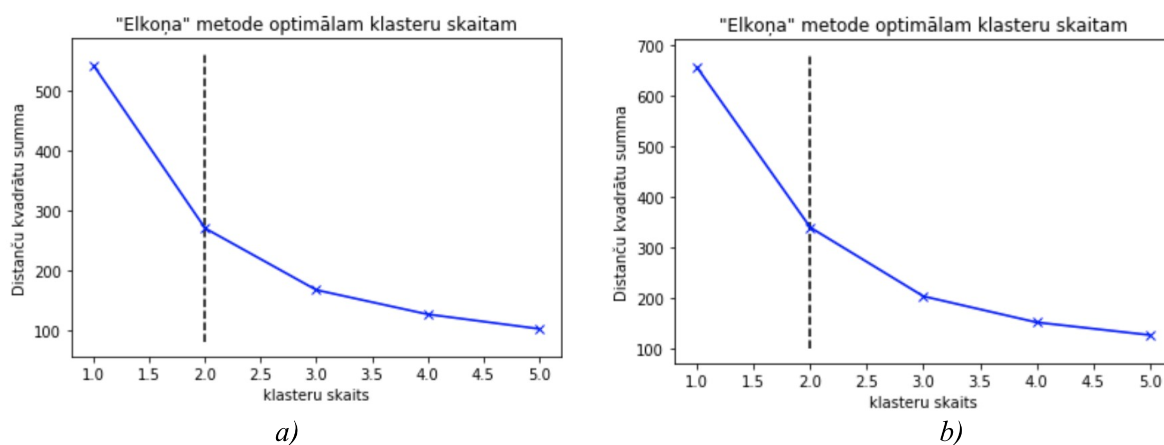
Iegūtos datus apstrādā ar WEKA, izmantojot klasterizāciju ar SimpleKMeans, kas ir k-vidējo implementācija WEKA. Šeit k-vidējam algoritmam ir zināms klasteru skaits. Izņemot brauciena vidējos(piem. brauciena ilgums) un ilgtermiņa vidējos atribūtus (piem. vidējais degvielas patēriņš) un izvēloties 4 labākos pēc analīzes ar InfoGainAttributeEval. Izdevās uzlabot nepareizi prognozēto instanču daudzumu no 40.6% līdz 39.5%. Pierādās tas pats rezultāts, kas analizējot ar uzraudzītām metodēm, ka izmantojot tikai labākos atribūtus, rezultāts uzlabojas. Tāpēc izmantosim tikai 4 labākos atribūtus tālākajā analīzē un paskatīsimies vai būs iespējams klasificēt datus nezinot braucēju sakaitu. Pievienojot vidējās vērtības pa 2,3,4 sekundēm un iepriekšējo periodu vērtības deva sliktāku prognozēšanas rezultātus. Labākie atribūti attēloti tabulā nr. 4.4. Atribūtu saraksts līdzīgs tam, kas iegūts pirmajā pētījuma daļā. Ir nelielas atšķirības, dēļ dažādiem sensoriem. Iespējams dēļ dzinēju degvielas tipa atšķirībām.

4.4. tabula

Labākie atribūti pēc InfoGainAttributeEval

Atribūts	**Apraksts**
Throttle_Position(Manifold)(prc)	Parāda par cik procentiem degvielas droselis ir atvērts pret maksimālo pozīciju.
Engine_Load(prc)	Norādītais griezes moments % no maksimālā
O2_Volts_Bank_1_sensor_2(V)	Skābekļa sensors uz gaisa/degvielas sajaukumu
Turbo_Boost_&_Vacuum_Gauge(bar)	Spiediens gaisa padevei uz dzinēju (korelē ar MAF)

Attēls 4.1 parāda aprēķināto distanču summu izmaiņu attiecībā pret padoto klasteru skaitu k-vidējam modelim. Kreisajā pusē ir redzams rezultāts datiem ar 2 klasteriem un labajā pusē 4 klasteriem. Vertikālā līnija parāda "Elkoņa" metodes atrasto klasteru skaitu.



a) *b)*

4.1. att. **Elkoņa metodes vizualizācija.**
a) dati ari 2 klasteriem, b) dati ar 4 klasteriem

Tabulā nr. 4.5. ir attēloti iegūtie rezultāti no k-vidējo algoritma kopā ar Elkoņa metodi. Cik atļāva dati, ir veikti vairāki mērījumi vienam klasteru skaitam. Kā redzams no tabulas, rezultāti ir visai slikti, gan klasteru skaits, gan ARI atzīme, kas mēra klasterizācijas pareizību pret patieso.

4.5 tabula

K-vidējo un Elkoņa metodes rezultāti

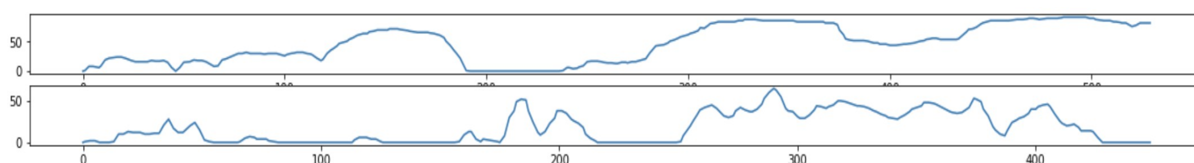
Reāls braucēju skaits	Modeļa prognozētais	ARI atzīme
1	2	0
1	2	0
2	2	0.00507
2	2	0.00821
2	2	0.08032
3	2	0.03996
3	2	0.01414
3	2	0.03065
4	2	0.02078

Kā redzams no rezultātiem, šī metode neparādīja labu rezultātu uz OBD2 datiem, lai identificētu braucēju neuzraudzītā veidā.

4.2.2. U-shapelet

Izstrādātais algoritms paredz klasificēt vadītājus pēc laika rindām. Tā kā esošā u-shapelet implementācija neparedz daudzdimensionālu laika rindu salīdzināšanu, bija nepieciešams izvēlēties atribūtu, kura laika rindu izmantot. Algoritma darbināšana aizņem krietnu laiku. Piemēram, lai izskaitļotu rezultātu 2 braucējiem, kas ir veikuši katrs pa 2 braucieniem aptuveni katrs ap 15 min ilgs, algoritmam nepieciešama apmēram 1 stunda laika. Laika ierobežojums neļāva izmēģināt visus atribūtus, tāpēc mēģinājumi tika sākti ar ātruma atribūtu, kas pēc loģikas un vizuālā attēlojuma izskatījās, ka varētu atbilst labam kandidātam.

Ātruma atkarības no laika vizualizācija 2 braucējiem ir attēlota attēlā nr. 4.2. Kā redzams līkne ir diezgan nehomogēna un plūstoša, kas varētu liecināt par labu kandidātu u-shapelet analīzei.



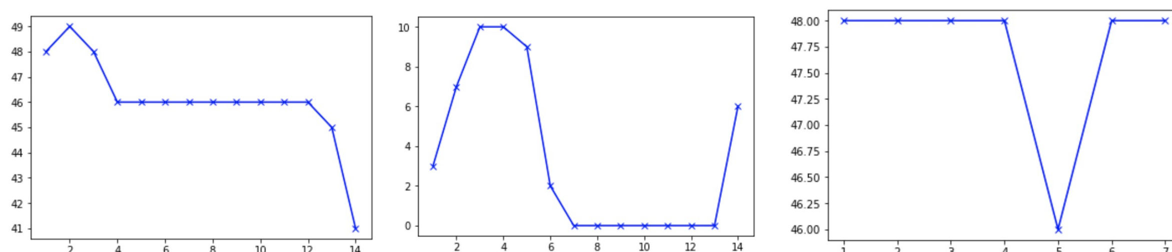
4.2. att. **Ātruma atkarība no laika 2 braucējiem.**
Uz vertikālās ass attēlots ātrums km/h, uz horizontālās ass sekundes no brauciena sākuma.

Klasifikācijas rezultāti dažādam braucēju(klasteru) skaitam ir apkopots tabulā 4.6. Kur ARI – Adjusted Rand Index, kas mēra klasificējuma tuvību, CRI – Change in Rand Index, ar kura palīdzību, izmantojot Rans Indeksa starpību, starp iepriekšējo un nākamo, pievienojot jaunu u-shapeletu, nosaka, kurā brīdī, nav jēga pievienot jaunu u-shapeletu, jo tas vairs neuzlabo klasifikāciju. Autori piedāvā šādu risinājumu, lai paātrinātu algoritma darbību, ja ir sarēķināti ļoti daudz u-shapeletu. No manas pieredzes, tas nebūt nav ilgākais algoritma posms, daudz ilgāk notiek u-shapeletu meklēšanas process. No 4.6. tabulas rezultātiem vadoties var redzēt, ka izmantojot visus u-shapeletus klasteru skaits ir precīzāks.

4.6. tabula

U-shapelet klasifikācijas rezultāti izmantojot ātruma atribūtu.

Braucēju skaits	U-shapeletu skaits	Visas u-shapelet distances		CRI izvēlētas distances	
		klasteri	ARI	klasteri	ARI
4	3	4	-0.035	2	-0.015
4	4	4	-0.035	2	-0.015
3	5	3	-0.103	3	-0.103
3	6	3	-0.103	3	-0.103
2	4	2	0	2	-0.084
2	4	2	0	2	0
2	3	2	-0.061	2	-0.149
1	2	2	0	2	0



4.3. att. **Atrastie 3 u-shapeleti 2 vadītāju klasifikācijai.**
Vertikālā ass ir ātrums km/h, vertikālā ass – sekundes.

Attēls 4.3. ir ilustratīvs piemērs, kā izskatās 3 atrastie u-shapeleti 2 braucējiem. Kā redzams no tabulas rezultātiem, izmantojot visu algoritma atrasto u-shapeletu distances ir iespējams precīzi noteikt klasteru skaitu apskatītajiem datiem. Var redzēt, ka, laižot algoritmu atkārtoti vienam klasteru skaitam, sākuma laika rinda patvaļīgi tiek izvēlēta cita un tiek atrasts cits skaits u-shapeletu, bet klasteru skaits tik un tā tiek uzminēts pareizs. Diemžēl klasteru sadalījums neatbilst īstenībai, ko var redzēt pēc ARI indeksa, kas ir ļoti zems. Indekss tuvu nullei parāda, ka līdzība ir ļoti maza, pie perfektas sakritības indekss ir 1. Tātad ar algoritma palīdzību var noteikt braucēju skaitu, bet nevar noteikt to sadalījumu pa braucieniem.

Vēl viena problēma, kas palika neatrisināta ir identificēt 1 braucēju. Diemžēl izmantotā "Elkoņa" metode nespēj izskaitīt distanču starpību starp 0 un 1 klasteru, tāpēc, ja reālais klasteris ir 1, labāko distanču starpība, ko atrod algoritms ir pie 2 klasteriem. Risinājumu šai problēmai ir piedāvājis Tibshirani et.al (2000) darbā "Estimating the number of clusters in a data set via the gap statistic", viņa piedāvātā metode "gap statistic" teorētiski spēj risināt problēmu, lai identificēt, ka klasteru skaits ir 1, tas ir datu kopa ir homogēna.[8] Šī darba ietvaros nav izdevies implementēt šo modeli, taču tas varētu būt labs papildinājums turpmākajos pētījumos.

Lai arī braukšanas ātruma atribūts parādīja nesliktu rezultātu klasteru skaita identificēšanā, turpmāk būtu nepieciešams izpētīt arī citu atribūtu laika rindas. Iespējams, ka ir labāki kandidāti. Piemēram tie atribūti, kas tika izvirzīti, kā svarīgākie analizējot ar InfoGainAttributeEval metodi. Vēl interesanti būtu padomāt, kā var apvienot atribūtu rezultātus precīzākai klasificēšanai. Šī brīža algoritms darbojas tikai uz katru atribūtu atsevišķi. Algoritms ir ļoti laikietilpīgs skaitļošanas ziņā, 4 braucēju analīze var aizņemt 4 stundas, darbinot uz 2.3 GHz Intel Core i5 procesora, tāpēc analizēt daudzus atribūtus ir diezgan laikietilpīgs process.

Ja runā par datiem, tad iespējams, ierakstīšanas intervāls 1 sekunde ir pārāk garš priekš automobiļa vadīšanas datiem, jo darbība notiek dinamiskā vidē. Ir iespējams, ka interesantākie shapeleti netiek atklāti dēļ intervāla lieluma.

SECINĀJUMI

Šī darba mērķis bija noskaidrot vai ir iespējams izmantojot tikai OBD2 datus savāktus no viena automobiļa, kuru vadīja dažādi braucēji, izmantojot datizraces algoritmus identificēt braucēju. Autovadītāja identificēšanas iespējas tiek pētītas izmantojot uzraudzītās un neuzraudzītas datizraces metodes.

No uzraudzītām metodēm autovadītāja klasificēšanai tika izmantoti 3 datizraces algoritmi: J48, Random Forest un KNN, no kuriem visefektīvākais bija Random Forest. Pētījums parādīja, ka, lai labāk noteiktu autovadītāju, nav nepieciešams izmantot visus iespējamos OBD2 rādītājus, bet gan ir jānovērtē atribūti pēc to informatīvā svarīguma klasifikācijai. Izmantojot tikai labākos atribūtus modeļi strādā gan ātrāk, gan precīzāk nosaka braucēju no datiem. Tas visdrīzāk ir saistīts ar modeļu pārākpielāgotību izmantojot daudz parametrus. Izmantojot OBD2 datus, kur personas ir vadījuši transportlīdzekli dažādos laikos un apstākļos pa patvaļīgiem maršrutiem, ir iespējams ar 91% varbūtību noteikt autovadītāju, kas atradās pie stūres.

Kā neuzraudzītā metode tika izvēlēta laika rindu analīze, konkrēti braukšanas ātruma analīze laikā, izmantojot u-shapelet algoritmu savienojumā ar k-vidējo un "elkoņa" metode klasteru meklēšanai. Apskatītajai datu kopai izdevās ar 100% precizitāti noteikt braucēju skaitu tikai analizējot braucienu datus, ar nosacījumu, ka braucēju skaits ir bijis 2 un vairāk. Ja braucējs ir bijis tikai 1, algoritms paredz, ka to ir bijis 2. Tas saistīts ar "elkoņa" metodes nepilnību spēt identificēt mazāk kā 2 klasterus. Šo problēmu potenciāli var risināt "elkoņa" metodes vietā izmantojot Tibshirani et.al (2000) piedāvāto "gap-statistic" metodi.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Byung Il Kwak, Jiyoung Woo and Huy Kang Kim, "*Know Your Master: Driver Profiling-based Anti-theft Method*", PST (Privacy, Security and Trust), 2016.

[2] "CAN bus" Pieejams: https://en.wikipedia.org/wiki/CAN_bus [Skatīts: 20-01-2019].

[3] Miro Enev, Alex Takakuwa, Karl Koscher, and Tadayoshi Kohno, *"Automobile Driver Fingerprinting"*, Proceedings on Privacy Enhancing Technologies ; 2016 (1):34–51

[4]F. Martinelli et al., *"Human behavior characterization for driving style recognition in vehicle system"*, Computers and Electrical Engineering (2018), https://doi.org/10.1016/j.compeleceng.2017.12.050

[5] "On-board diagnostics" Pieejams: https://en.wikipedia.org/wiki/On-board_diagnostics [Skatīts: 20-01-2019].

[6] WEKA. Pieejams: https://www.cs.waikato.ac.nz/ml/weka/ [Skatīts: 20-01-2019].

[7] Zaldivar, Jorge & Calafate, Carlos & Cano, Juan-Carlos & Manzoni, Pietro. (2011). *"Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones."*, Proceedings - Conference on Local Computer Networks, LCN. 813-819. 10.1109/LCN.2011.6115556.

[8] Tibshirani R., Walther G., Hastie T. (2000) "Estimating the number of clusters in a data set via the gap statistic". R. Statist. Soc. B (2001) 63, Part 2, pp. 411-423.

[9] "InfoGainAttributeEval" Pieejams: https://wiki.pentaho.com/display/DATAMINING/InfoGainAttributeEval [Skatīts: 20-01-2019].

[10] "OBD-II datasets". Pieejams: https://www.kaggle.com/cephasax/obdii-ds3 [Skatīts: 20-04-2019].

[11] "OBD-II PIDs". Pieejams: https://en.wikipedia.org/wiki/OBD-II_PIDs [Skatīts: 20-04-2019].

[12] Zakaria J., Mueen A., Keogh E. *"Clustering Time Series using Unsupervised-Shapelets"*. 2012 IEEE 12th International Conference on Data Mining, 2012

[13] Podnieks K., "Datizraces algoritmi: klasteru atklāšana", Latvijas Universitātes, Datizraces kursa prezentācija. Pieejams: http://podnieks.id.lv/slides/mining/metodes1%20CLUST.pdf [Skatīts: 20-04-2019].

[14] "sklearn.metrics.pairwise.euclidean_distances". Pieejams: https://scikit-

learn.org/stable/modules/generated/sklearn.metrics.pairwise.euclidean_distances.html [Skatīts: 20-04-2019].

[15] "k-means clustering". Pieejams: https://en.wikipedia.org/wiki/K-means_clustering [Skatīts: 20-04-2019].

[16] Liao, T. Warren. (2005) *"Clustering of time series data—a survey"*. Pattern Recognition 38 (2005) 1857–1874.

[17] "sklearn.cluster.KMeans". Pieejams: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html [Skatīts: 20-05-2019].

[18] "kneed". Pieejams: https://github.com/arvkevi/kneed [Skatīts: 20-05-2019].

[19] Satopaa V., Albrecht J., Irwin D., Raghavan B. (2007) *"Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior"*. Third IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC 2007)

[20] Langkvist M., Karlsson L., Loutfi A. (2013) *"A review of unsupervised feature learning and deep learning for time-series modeling"*. Pattern Recognition Letters 42 (2014) 11–24

[21] Shalabi L. Al, Shaaban Z., Kasasbeh B. (2006) *"Data Mining: A Preprocessing Engine"*. Journal of Computer Science 2 (9): 735-739, 2006.

[22] "adjusted_rand_score". Pieejams: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html [Skatīts: 20-05-2019].

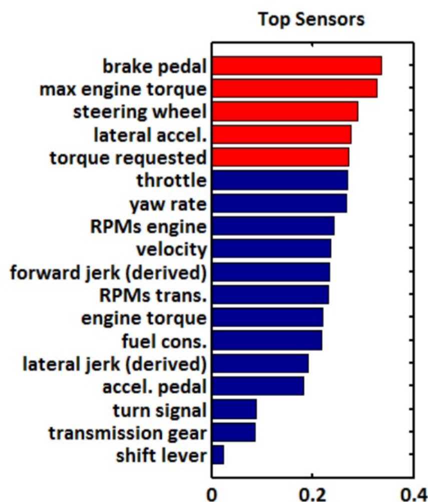
PIELIKUMI

Pielikums nr. 1

Sensor	Control Module	Range	Update Rate	Summary
Brake Pedal Position	Brake	$0-100\%$	15ms	Degree to which driver is depressing the brake pedal.
Steering Wheel Angle	Brake	$-2048-2048°$	20ms	Positive when steering wheel is rotated counterclockwise.
Lateral Acceleration	Brake	$-32-32\frac{deg}{sec}^2$	25ms	Measurement from an accelerometer, positive in left direction.
Yaw Rate	Brake	$-128-128\frac{deg}{sec}$	30ms	Vehicle rotation around vertical axis, positive in left turn.
Gear Shift Lever	Transmission	$1-6$	50ms	An indication of the state of the transmission shift lever position as selected by the driver.
Vehicle Speed	Transmission	$0-317.46\frac{miles}{hr}$	100ms	Vehicle speed computed using the angular velocity of the primary (high torque) axle.
Estimated Gear	Transmission	$1-6$	60ms	An estimate of the gear that the transmission has achieved (will not change its value until a shift is complete).
Shaft Angular Velocity	Transmission	$0-16383.8rpm$	25ms	Speed of the transmission output shaft; on front wheel drive configurations this signal represents the average speed of the front axles.
Accelerator Pedal Position	Engine	0-100%	20ms	Degree to which driver is depressing the accelerator pedal.
Engine Speed (RPMs)	Engine	$0-16383.8rpm$	15ms	High-resolution engine speed in revolutions per minute.
Driver Requested Torque	Engine	$-848-1199Nm$	60ms	Value is based on the acceleration and brake pedal characteristics.
Maximum Engine Torque	Engine	$-848-1199Nm$	125ms	This signal is the calculated maximum torque that the engine can provide under the current circumstances (altitude, temperature, etc.), based on wide-open throttle conditions.
Fuel Consumption Rate	Engine	$0-102\ \frac{liters}{hr}$	125ms	Instantaneous fuel consumption rate computed based on the average over the last sample period (e.g., 100 ms).
Throttle Position	Engine	$0-100\%$	30ms	Zero represents the near closed bore position (idle, coast) and 100% represents full available power.
Turn Signal	N/A	$2/1/0$	N/A	Off, left, or right turn signal.

List of sensors used in analysis. Note that ranges are based on sensor hardware and may not necessarily reflect the empirical levels reachable during normal operation.

[3]



[3]

Pielikums nr. 2.

Feature	Type of vehicle data	Range	Description	Feature rank
Long term fuel trim bank1	Fuel	-100 – 100 (%)	The correction value being used by the fuel control system in loop modes of operation.	1
Intake air pressure	Fuel	0 – 255 (kPA)	A pressure of air inhaled to engine.	4
Accelerator Pedal value	Fuel	0 – 100 (%)	The degree to which driver is depressing the accelerator pedal.	9
Fuel consumption	Fuel	0 – 10000 (mcc)	The fuel efficiency of an engine.	11
Friction torque	Engine	0 – 100 (%)	A torque caused by the frictional force that occurs.	3
Maximum indicated engine torque	Engine	0 – 100 (%)	A calculated value of maximum torque.	5
Engine torque	Engine	0 – 100 (%)	A force that is spinning engine crankshaft.	6
Calculated load value	Engine	0 – 100 (%)	A percentage of peak available torque.	7
Activation of Air compressor	Engine	0 or 1	A value of air compressor's working.	8
Engine coolant temperature	Engine	-40 – 215 (°C)	The temperature of the engine coolant of an internal combustion engine.	10
Transmission oil temperature	Transmission	-40 – 215 (°C)	A fluid temperature inside the transmission.	2
Wheel velocity, front, left-hand	Transmission	0 – 511.75 (km/h)	The speed of the left front wheel.	12
Wheel velocity, front, right-hand	Transmission	0 – 511.75 (km/h)	The speed of the right front wheel.	14
Wheel velocity, rear, left-hand	Transmission	0 – 511.75 (km/h)	The speed of the left rear wheel.	13
Torque converter speed	Transmission	0 – 16383.75 (rpm)	A particular kind of fluid coupling that is used to transfer rotating power from a prime mover.	15

[1]

Pielikums nr. 3.

#	**Feature**
5	*Intake_air_pressure*
8	*Engine_soacking_time*
12	*Long_Term_Fuel_Trim_Bank1*
15	*Torque_of_friction*
35	*Transmission_oil_temperature*
50	*Steering_wheel_speed*

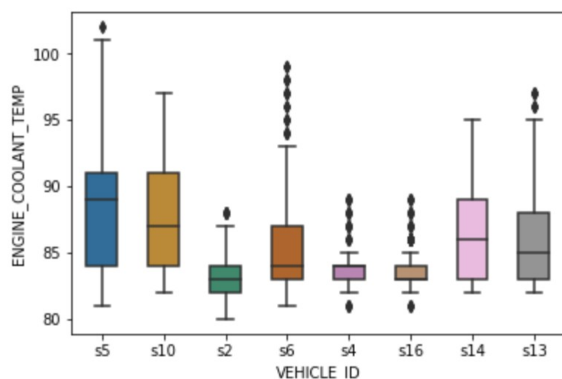
Nozīmīgākie parametri. [4]

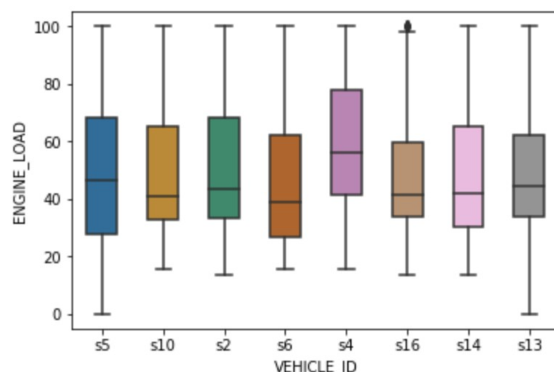
Pielikums nr. 4.

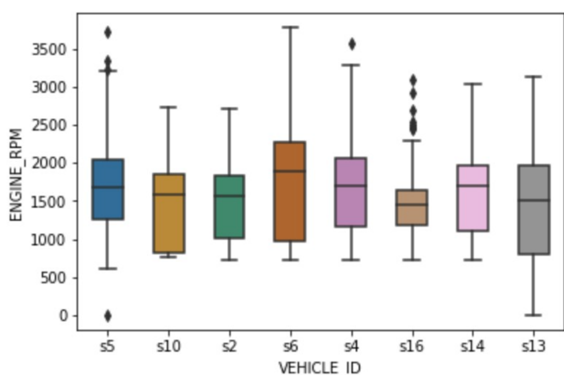
Sākotnējo datu apraksts. Apraksts ņemts no līdzīgiem darbiem.[1][3][4]

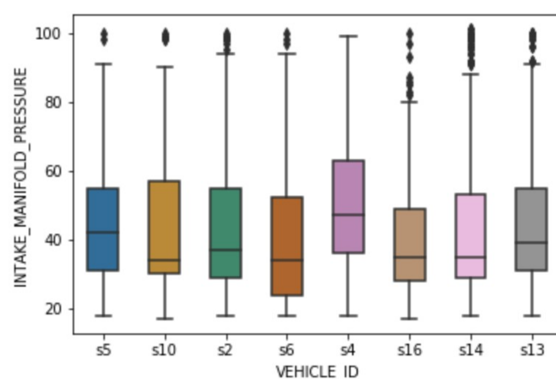
Nosaukums	Tulkojums	Mērvienība	Apraksts
VEHICLE_ID	Braucēja ID	teksts	Braucēja identifikators (s1, s2 ...)
BAROMETRIC_PRESSURE	Atmosfēras spiediens	kPa	Ārējais atmosfēras spiediens
ENGINE_COOLANT_TEMP	Dzesētāja temperatūra	C	Dzinēja dzesēšanas šķidruma temperatūra
ENGINE_LOAD	Dzinēja slodze	%	Norādītais griezes moments % no maksimālā
ENGINE_RPM	Dzinēja apgriezieni	RPM	Dzinēja apgriezieni, kas tiek mērīti kā apgriezienu skaits minūtē
INTAKE_MANIFOLD_PRESSURE	Ienākošā gaisa spiediens	kPa	Tiek mērīts ar MAP sensoru. Mēra spiedienu gaisam, kas tiek padots uz iekšdedzes dzinēju, lai noteiktu gaisa blīvumu.
MAF	MAF	g/s	Mēra gaisa daudzumu, kas tiek padots uz dzinēju.
AIR_INTAKE_TEMP	Ienākošā gaisa temp.	C	Dzinējam padotā gaisa temperatūra
SPEED	Ātrums	km/h	Transportlīdzekļa ātrums
Short Term Fuel Trim Bank 1	Īstermiņa degvielas korekcija	%	Degvielas korekcija, kas tiek padota uz dzinēju. Atkarīga no gais mērījumiem, lai veidotos pareiza sakaukuma proporcija.
ENGINE_RUNTIME	Dzinēja laiks	'00:00:00'	Laiks, cik strādā dzinējs.
THROTTLE_POS	Droseļa pozīcija	%	Parāda par cik procentiem degvielas droselis ir atvērts pret maksimālo pozīciju.
TIMING_ADVANCE	Laika korekcija	%	Var būt saistīta ar dzirksteles momenta vai droseļa korekciju labākai degvielas maisījuma sadegšanai.

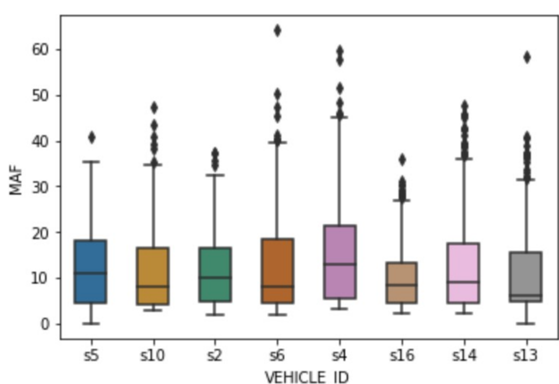
Pielikums nr. 5.

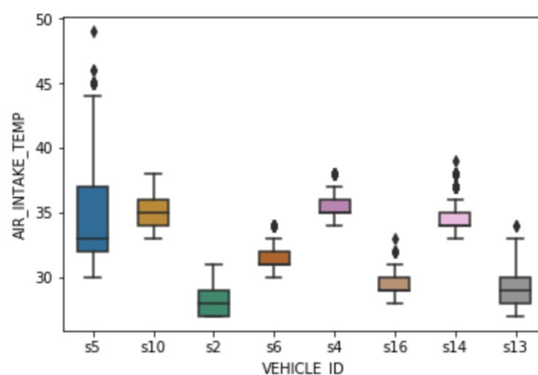


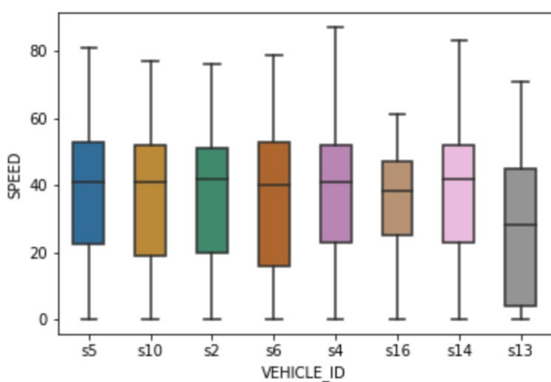


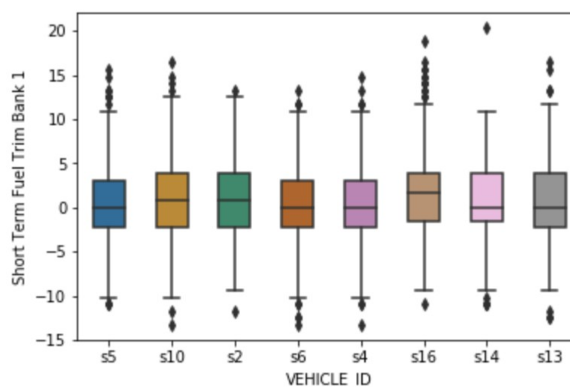


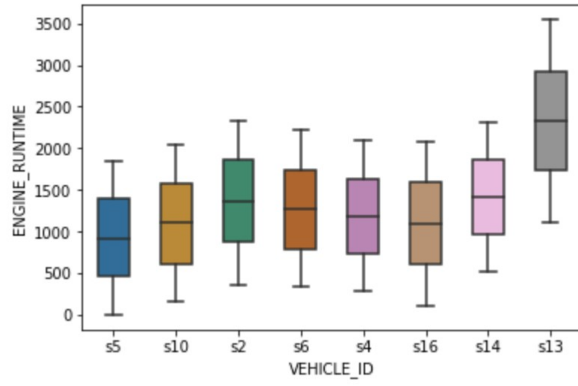




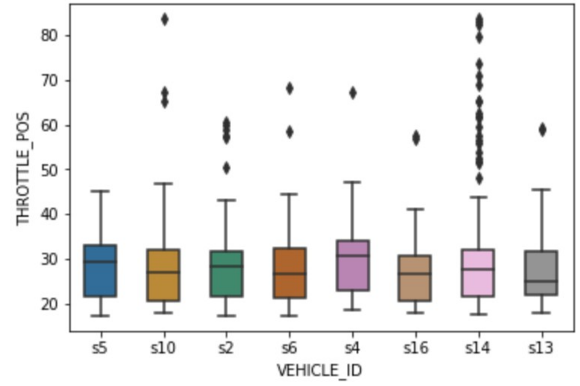




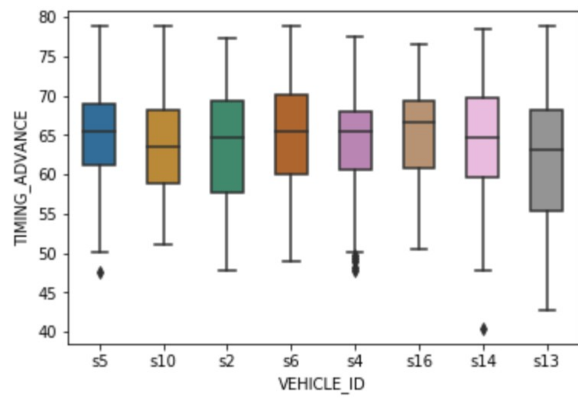




ENGINE_RUNTIME
3500
3000
2500
2000
1500
1000
500
0
s5
s10
s2
s6
s4
s16
s14
s13
VEHICLE_ID



THROTTLE_POS
80
70
60
50
40
30
20
s5
s10
s2
s6
s4
s16
s14
s13
VEHICLE_ID



TIMING_ADVANCE
80
75
70
65
60
55
50
45
40
s5
s10
s2
s6
s4
s16
s14
s13
VEHICLE_ID

Pielikums nr. 6.

J48

Scheme: weka.classifiers.trees.J48 -C 0.15 -M 2

Number of Leaves : 236

Size of the tree : 471

Time taken to build model: 0.06 seconds

Correctly Classified Instances 2973 85.6279 %

Incorrectly Classified Instances 499 14.3721 %

RandomForest

Scheme: weka.classifiers.trees.RandomForest -P 100 -I 200 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S -1

Bagging with 200 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S -1 -do-not-check-capabilities

Time taken to build model: 1.91 seconds

Correctly Classified Instances 3096 89.1705 %

Incorrectly Classified Instances 376 10.8295 %

KNN

Scheme: weka.classifiers.lazy.IBk -K 3 -W 0 -I -A Scheme: weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.FilteredDistance -R first-last -F \\\"weka.filters.unsupervised.attribute.RandomProjection -N 10 -R 42 -D Sparse1\\\" -D \\\"weka.core.EuclideanDistance -R first-last\\\"\""

Time taken to build model: 0.01 seconds

Correctly Classified Instances 2727 78.5426 %

Incorrectly Classified Instances 745 21.4574 %

Pielikums nr. 7.

J48

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2

Attributes: 6

ENGINE_COOLANT_TEMP

ENGINE_RPM

MAF

AIR_INTAKE_TEMP

TIMING_ADVANCE

VEHICLE_ID

Number of Leaves : 238

Size of the tree : 475

Time taken to build model: 0.04 seconds

Correctly Classified Instances 2993 86.2039 %

Incorrectly Classified Instances 479 13.7961 %

Random Forest

Scheme: weka.classifiers.trees.RandomForest -P 100 -I 200 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S -1

Attributes: 6

ENGINE_COOLANT_TEMP

ENGINE_RPM

MAF

AIR_INTAKE_TEMP

TIMING_ADVANCE

VEHICLE_ID

Correctly Classified Instances 3146 90.6106 %

Incorrectly Classified Instances 326 9.3894 %

KNN

Scheme: weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.FilteredDistance -R first-last -F \\\"weka.filters.unsupervised.attribute.RandomProjection -N 10 -R 42 -D Sparse1\\\" -D \\\"weka.core.EuclideanDistance -R first-last\\\"\""

Attributes: 7

ENGINE_COOLANT_TEMP

ENGINE_RPM

MAF

AIR_INTAKE_TEMP

ENGINE_RUNTIME

TIMING_ADVANCE

VEHICLE_ID

Time taken to build model: 0 seconds

Correctly Classified Instances 2901 83.5541 %

Incorrectly Classified Instances 571 16.4459 %

Pielikums nr. 8.

```
# import matplotlib
#!/usr/bin/env python
# coding: utf-8
from __future__ import division
import matplotlib.pyplot as plt
import numpy as np
import dataFunc
import pandas as pd
import ast
import os
from sklearn import preprocessing
from scipy import stats
from sklearn.metrics.pairwise import euclidean_distances
from scipy.spatial.distance import euclidean
import time
import datetime
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
from kneed import KneeLocator
import random

#read files and process
############################################################
#read data files from folder to dictionary of data frames
path = '../driving data/colt-all-atributes/4drivers/'
names = ['.csv']
dfAll = []
files = []
counter = 0
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
    for file in f:
        if '.csv' in file:
            if '~' in file:
                continue
            df_temp = ''
            counter += 1
            name = file.split('-')[0]
            filepath = r + file
            df_temp = pd.read_csv(filepath, delimiter = ',', na_values='-')
            df_temp['driver'] = str(name)
#           print(name + ' ' + str(df_temp.shape) + ' ' + file)
#           print(filepath)
            print(df_temp.shape)
            dfAll.append(df_temp.reset_index(drop=True))

############################################################
#read columne list to select from file
col_list = list()
fileName = 'col_list-IG.txt'
with open(fileName) as f:
    for line in f:
        print(line.strip('\n'))
        col_list.append(line.strip('\n'))
```

```
############################################################
#clean from rows with column names
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i][dfAll[i].iloc[:,0] != dfAll[i].columns[0]]
# removo non acii from column names
for i in range(len(dfAll)):
    dataFunc.remove_non_ascii_column(dfAll[i])
#select only columns from the list
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i][(col_list)]
# replace infinity with 0
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i].replace('∞',0).round(3)
#values to numeric except 'driver'
for i in range(len(dfAll)):
    dataFunc.convert_numeric(dfAll[i],)
#fill NaN values with 0
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i].fillna(0)

#give concat data frame to function
def create_classes(df):
    cls_actual = []
    df = df.reset_index(drop=True)
    for i in range(len(df.index)):
        cls_actual.append(df['driver'][i])

    cls_names = set(cls_actual)
    cls_names = list(cls_names)

    map_name = {}
    class_count = 0
    for name in cls_names:
        print(name + ' = ' + str(class_count))
        map_name[name] = class_count
        class_count += 1

    for i in range(len(cls_actual)):
        cls_actual[i] = map_name[cls_actual[i]]

#     print(cls_actual)
    return cls_actual

#concat files after cleaning and making class vector
df_concat_with_driver = pd.concat(dfAll)
df_concat_with_driver.shape
#DROP driver column
df_concat = df_concat_with_driver.drop(columns=['driver'])
df_concat_with_driver = df_concat_with_driver.reset_index(drop=True)
# df_concat = df_concat.drop(columns=['Trip_Time(Since_journey_start)(s)']).reset_index(drop=True)

#normalize with MinMaxScaler
col_list_1 = col_list
col_list_1.remove('driver')
mms = preprocessing.MinMaxScaler()
mms.fit(df_concat)
data_transformed = mms.transform(df_concat)
# df_transformed = pd.DataFrame(data_transformed,columns = col_list_1)
# df_transformed['driver'] = df_concat_with_driver['driver']
```

```
def clusterDIS(DIS, numberOfClusters):

    sumDIS = float('inf')
    #make coupe of iterations to find best kmeans prediction
    for s in range(10):
        kmeans = KMeans(n_clusters=numberOfClusters).fit(DIS)
        SUMD = kmeans.inertia_
#          print('sumdis' + str(SUMD))
        if SUMD < sumDIS :
            sumDIS = SUMD
            cls = kmeans.labels_
    return cls,sumDIS

#make coupel of iterations to find best kmeans prediction
Kmin = 1
Kmax = int(len(dfAll) / 2) + 2
# Kmax = 6
CLS = []
SUMDIS = []
K = []

for k in range(Kmin, Kmax):
    print(k)
    cls,sumDIS = clusterDIS(data_transformed, k)
    K.append(k)
    CLS.append(cls)
    SUMDIS.append(sumDIS)


#####################################################
# . PLOT AND FIND CLUSTERS


plt.plot(K, SUMDIS, 'bx-')
plt.xlabel('klasteru skaits')
plt.ylabel(u'Distanču kvadrātu summa')
plt.title(u'"Elkoņa" metode optimālam klasteru skaitam')
plt.vlines(clusters, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.show()

kn = KneeLocator(K, SUMDIS, curve='convex', direction='decreasing')
clusters = kn.knee
# print('Clusters all : ' + str(clusters))
# print('Labesl all : ' + str(CLS[K.index(clusters)]))


#####################################################
# . ARI RESULTS

cls_true = create_classes(df_concat_with_driver)
cls_all = CLS[K.index(clusters)]
# print('Actual  : ' +'[' + str(' '.join(map(str, cls_true))) + ']')
# print('cls_all : ' + str(cls_all))
print('Aprēķinātais klasteru skaits : ' + str(clusters))

Rscore_all = round(adjusted_rand_score(cls_all, cls_true),5)
print('ARI : ' + str(Rscore_all))
```

Pielikums nr. 9.

```
#!/usr/bin/env python
# coding: utf-8
# import matplotlib
from __future__ import division
import matplotlib.pyplot as plt
import numpy as np
import dataFunc
import pandas as pd
import ast
import os
from sklearn import preprocessing
from scipy import stats
from sklearn.metrics.pairwise import euclidean_distances
from scipy.spatial.distance import euclidean
import time
import datetime
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
from kneed import KneeLocator
import random
#read files and process
############################################################
#read data files from folder to dictionary of data frames
path = '../driving data/colt-all-atributes/4drivers/'
names = ['.csv']
dfAll = []
files = []
counter = 0
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
    for file in f:
        if '.csv' in file:
            if '~' in file:
                continue
            df_temp = ''
            counter += 1
            name = file.split('-')[0]
            filepath = r + file
            df_temp = pd.read_csv(filepath, delimiter = ',', na_values='-')
            df_temp['driver'] = str(name)
#           print(name + ' ' + str(df_temp.shape) + ' ' + file)
#           print(filepath)
            print(df_temp.shape)
            dfAll.append(df_temp)
############################################################
#read columne list to select from file
col_list = list()
fileName = 'list-my.txt'
with open(fileName) as f:
    for line in f:
        print(line.strip('\n'))
        col_list.append(line.strip('\n'))
############################################################
#clean from rows with column names
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i][dfAll[i].iloc[:,0] != dfAll[i].columns[0]]
```

```
# remove non acii from column names
for i in range(len(dfAll)):
    dataFunc.remove_non_ascii_column(dfAll[i])
#select only columns from the list
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i][(col_list)]
# replace infinity with 0
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i].replace('∞',0).round(3)
#values to numeric except 'driver'
for i in range(len(dfAll)):
    dataFunc.convert_numeric(dfAll[i],)
#fill NaN values with 0
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i].fillna(0)
#re index
for i in range(len(dfAll)):
    dfAll[i] = dfAll[i].reset_index(drop=True)
#divide to smaller Data Frames
observations = 300
df_temp = []
for i in range(len(dfAll)):
    idx = len(dfAll[i].index)
    n = round(idx / observations , 0)
#     print(idx)
#     print(n)
    if n > 1:
        print(n)
        md = int(idx%n)
        print(md)
        if md != 0:
            for d in range(md):
                print('drop index ' + str(d))
                dfAll[i] = dfAll[i].drop(d)
        dfs = np.split(dfAll[i], n, axis=0)
        for j in range(len(dfs)):
            df_temp.append(dfs[j])
    else:
        df_temp.append(dfAll[i])

dfAll = df_temp

for ii in range(len(dfAll)):
    dfAll[ii] = dfAll[ii].reset_index(drop=True)
    print(dfAll[ii].shape)

#visualise data
fig = plt.figure(figsize=(20,20))
for i in range(len(dfAll)):
    name = dfAll[i]['driver'][1]
    ax=plt.subplot(20,1,i+1)
        dfAll[i].plot(kind='line',y='Speed_(OBD)(km/h)',x='Trip_Time(Since_journey_start)(s)',   ax=ax,
legend=False).set_ylabel(name)
plt.show()

# In[19]:
def extractUshapelets(df_dic, col_to_check, minSh = 5, maxSh = 20):
    tss = time.time()
    start_time = datetime.datetime.fromtimestamp(tss).strftime('%Y%m%d_%H:%M')
    print('Process started : ' + str(start_time))
```

```
Sh_set = {}
ts_dic = []
for i in range(len(df_dic)):
    ts_dic.append(df_dic[i][col_to_check])

start_index = random.randint(0,len(df_dic)-1)
ts = ts_dic[start_index]
run_cnt = 0
print("Starting with index " + str(start_index))

while True:
    run_cnt += 1
    temp_shapelet = []
    temp_gap = []
    temp_distance = []

    if run_cnt > len(df_dic)/2:
        break

    print("Run " + str(run_cnt) + ". " +str(len(ts_dic))+ " time series in Dataset.")
    for sl in range(minSh, maxSh) :
        for i in  range(len(ts.index) - sl):
            shaplet = ts[i:(i+sl)]
            if min(shaplet) == max(shaplet):
                continue
            g, d = computeGap(shaplet, ts_dic)
            temp_shapelet.append(shaplet.tolist())
            temp_gap.append(g)
            temp_distance.append(d)

    if min(temp_gap) == max(temp_gap):
        i_gap = temp_gap[0]
        index1 = 0
        dt = temp_distance[0]
    else:
        index1 = temp_gap.index(max(temp_gap))
        i_gap = temp_gap[index1]
        dt = temp_distance[index1]

    print('gap : ' + str(i_gap))
    print('index : ' + str(index1))
    print('distance ' + str(dt))

    #if gap is 0 means that bad shapelet is chosen, move to searching for next.
    index2 = 0
    if i_gap == 0 :
        while True:
          index2 = random.randint(0,len(ts_dic)-1)
          if index2 != index1:
            break
        ts = ts_dic[index2]
        print('next index = ' +str(index2))
        continue

    Sh_set[run_cnt] = [i_gap, temp_shapelet[index1]]

    #get distance for best shapelet from TS
    dis = computeDistance(temp_shapelet[index1],ts_dic)
    print('best gap distances : '+ str(dis))
    print('marger dt distance : ' + str(dt))
```

```
        DA = {}
        for r in range(len(dis)):
            if dis[r] <= dt:
                DA[r] = dis[r]

        if len(DA) <= 1:
            print("Process ended after - if len(DA) <= 1:")
            break

        else:
            index2 = dis.index(max(dis))
            print('For next run, max distance time series index = ' + str(index2))
            ts = ts_dic[index2]
            DA_list=list(DA.values())
            treshold = np.mean(DA_list) + np.std(DA_list)

            for n in range(len(dis)-1,-1,-1):
                if dis[n] < treshold:
                    print( str(n) + '. Data set is dropped out after ' + str(run_cnt) + ' run.')
                    ts_dic.pop(n)

            if len(ts_dic) < 2 :
                print('process ended after - if len(ts_dic) < 2:')
                break

    tss = time.time()
    end_time = datetime.datetime.fromtimestamp(tss).strftime('%Y%m%d_%H:%M')
    print('Process ended : ' + str(end_time))
    return Sh_set

# In[20]:
def computeGap(shapelet, dic):
    dis = computeDistance(shapelet, dic)
    dis = sorted(dis)
    #number of clusters
    k = 2
    maxGap = 0
    dt = 0
    for i in range(len(dis)-1):
        DB = list()
        DA = list()
        d = round((dis[i] + dis[i+1])/2 , 4)
        for j in range(len(dis)):
            if dis[j] < d:
                DA.append(dis[j])
            else:
                DB.append(dis[j])
        if len(DA) > 1 and len(DB) > 1:
            #find maximum gap
            mA = np.mean(DA)
            mB = np.mean(DB)
            sdA = np.std(DA)
            sdB = np.std(DB)
            gap = round(mB - mA - sdA - sdB , 4)
#           print('gap : ' + str(gap))
            if gap > maxGap:
                maxGap = gap
                dt = d

    return maxGap, dt
```

```
# In[7]:
def computeDistance(shaplet, dic):
    dis = []
    sLen = len(shaplet)
    shaplet = np.nan_to_num(stats.zscore(shaplet))

    for i in range(len(dic)):
        dis.append(1000)
        ts = dic[i]
#        d = list()
        for j in range(len(ts)- sLen):
            z = ts[j:(j + sLen)]
            z = np.nan_to_num(stats.zscore(z))
            d = euclidean(z, shaplet)
            d = round(d,4)
#            d.append(euclidean(z, shaplet))
            if d < dis[i]:
                dis[i] = round((d / np.sqrt(sLen)),4)
#        print(dis[i])
    # add SQRT normalization to distances
    return dis

# In[21]:
def clusterData(TS_dataSet, ShapletSet, numberOfClusters):
    #shaplet need to be arranged starting with the biggest gap. as the greedy search is implemented.
    DIS = pd.DataFrame()
    cls_real = []
    for i in range(len(TS_dataSet)):
        cls_real.append(0)
    cls = {}
    Change_RI = {}
    sumDistanceList = []

    for l in range(len(ShapletSet)):
        shp = ShapletSet[l]
        dis = computeDistance(shp, TS_dataSet)
        DIS[l] = dis
        sumDIS = float("inf")
        #make coupe of iterations to find best kmeans prediction
        for s in range(10):
            kmeans = KMeans(n_clusters=numberOfClusters).fit(DIS)
            SUMD = kmeans.inertia_
            sumDistanceList.append(SUMD)
            if SUMD < sumDIS :
                sumDIS = SUMD
                cls[l] = kmeans.labels_

        if l == 0 :
            a = (adjusted_rand_score(cls[l], cls_real))
            Change_RI[l]=(1 - a)
        else:
            a = (adjusted_rand_score(cls[l], cls[l-1]))
            Change_RI[l]=(1 - a)

    maxIndex = max(Change_RI, key=Change_RI.get)
    clusterLabel = cls[maxIndex]
    SmD = sumDistanceList[maxIndex]

    return clusterLabel, SmD
```

```
# In[22]:
def clusterDIS(TS_dataSet, ShapletSet1, numberOfClusters):
   #shaplet need to be arranged starting with the biggest gap. as the greedy search is implemented.
   DIS = pd.DataFrame()
   #create map of vector distances for each shapelet
   for l in range(len(ShapletSet1)):
      shp = ShapletSet1[l]
      dis = computeDistance(shp, TS_dataSet)
      DIS[l] = dis
      sumDIS = float("inf")
   #make coupe of iterations to find best kmeans prediction
   for s in range(10):
      kmeans = KMeans(n_clusters=numberOfClusters).fit(DIS)
      SUMD = kmeans.inertia_
      if SUMD < sumDIS :
         sumDIS = SUMD
         cls = kmeans.labels_
   return cls,sumDIS

# In[23]:
def create_classes(df):
   cls_actual = []
   for i in range(len(df)):
      cls_actual.append(dfAll[i]['driver'][0])

   cls_names = set(cls_actual)
   cls_names = list(cls_names)

   map_name = {}
   class_count = 0
   for name in cls_names:
      print(name + ' = ' + str(class_count))
      map_name[name] = class_count
      class_count += 1

   for i in range(len(cls_actual)):
      cls_actual[i] = map_name[cls_actual[i]]

#     print(cls_actual)
   return cls_actual

#find best K
ShapletSet = extractUshapelets(dfAll,'Speed_(OBD)(km/h)',5,15)
print(ShapletSet)
os.system( "say U shapelet process has ended" )
#####################################################
# FIND CLUSTERS USING SHAPELETS AND DISTANCE VECOTRS
mmm = []
for key in ShapletSet:
   mmm.append(ShapletSet[key][1])

Kmin = 1
Kmax = int(round(len(dfAll) / 2,0)) + 1
tss = []
for i in range(len(dfAll)):
   tss.append(dfAll[i]['Speed_(OBD)(km/h)'])

CLS = []
SUMDIS = []
```

```
K = []

CLS_CRI = []
SUMDIS_CRI = []
K_CRI = []

for k in range(Kmin, Kmax):
    print(k)
    cls,sumDIS = clusterDIS(tss, mmm, k)
    K.append(k)
    CLS.append(cls)
    SUMDIS.append(sumDIS)

    cls2,sumDIS2 = clusterData(tss, mmm, k)
    K_CRI.append(k)
    CLS_CRI.append(cls2)
    SUMDIS_CRI.append(sumDIS2)

#####################################################
# . PLOT AND FIND CLUSTERS
kn = KneeLocator(K, SUMDIS, curve='convex', direction='decreasing')
clusters = kn.knee

plt.plot(K, SUMDIS, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k (All shapelets)')
plt.vlines(clusters, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.show()

print('Clusters all : ' + str(clusters))
print('Labesl all : ' + str(CLS[K.index(clusters)]))

kn2 = KneeLocator(K, SUMDIS_CRI, curve='convex', direction='decreasing')
clusters_cri = kn2.knee

plt.plot(K, SUMDIS, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k (CRI chosen shapelets)')
plt.vlines(clusters, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.show()
print('Clusters CRI : ' + str(clusters_cri))
print('Labesl CRI : ' + str(CLS_CRI[K_CRI.index(clusters_cri)]))
#####################################################
# . ARI RESULTS
cls_true = create_classes(dfAll)
cls_all = CLS[K.index(clusters)]
cls_CRI = CLS_CRI[K_CRI.index(clusters_cri)]
print('Actual  : ' +'[' + str(' '.join(map(str, cls_true))) + ']')
print('cls_all : ' + str(cls_all))
print('All number of clusters : ' + str(clusters))
print('cls_CRI : ' + str(cls_CRI))
print('CRI number of clusters : ' + str(clusters_cri))

Rscore_all = round(adjusted_rand_score(cls_all, cls_true),3)
Rscore_CRI = round(adjusted_rand_score(cls_CRI, cls_true),3)
print('ARscore_all : ' + str(Rscore_all))
print('ARscore_CRI : ' + str(Rscore_CRI))
```

Pielikums nr. 10.

```
#!/usr/bin/env python
# coding: utf-8
from __future__ import division
from datetime import timedelta
from datetime import datetime
from time import mktime
from unidecode import unidecode
from sklearn import preprocessing
import pandas as pd
import os
import numpy as np
import matplotlib as mpl
import time
import matplotlib.pyplot as plt
import seaborn as sns
import arff
import datetime
import time
from scipy import stats

#initialise all functions
#-----------------------------------------------------------------------
#print df shape function
def print_shape(df1):
    for i in range(len(df1)):
        print(str(i) + ' '+ str(df1[i].shape))
#-----------------------------------------------------------------------
# visualise how total row count changes to NaN value treshold to drop row.
def visualise_treshold(df1):
    x_ratio = []
    y_filtered = []
    y_ratio = 0.80
    for r in range(70,101,1):
        dfnew = []
        y_rows = 0
        y_ratio = r/100
        for i in range(len(df1)):
            y_tresh = len(df1[i].columns) * y_ratio
            y_rows += df1[i].dropna(axis=0, thresh=y_tresh).shape[0]
        x_ratio.append(y_ratio)
        y_filtered.append(y_rows)

    print(x_ratio)
    print(y_filtered)
    plt.plot(x_ratio,y_filtered)
    plt.ylabel('total rows')
    plt.xlabel('portion of noNAN')
    plt.show()
#-----------------------------------------------------------------------
# function - drop rows with lot of NA values and reindex
def dropNaNrows(df1,y_ratio=0.9):
    for i in range(len(df1)):
        y_tresh = len(df1[i].columns) * y_ratio
        df1[i] = df1[i].dropna(axis=0, thresh=y_tresh)
        df1[i] = df1[i].reset_index(drop=True)
#-----------------------------------------------------------------------
```

```
#drop all constants, but not label
def drop_const_column(df1, label='driver'):
    dfAll_no_const = []
    for i in range(len(df1)):
        df_temp = []
#        print(str(i) + str(df1[i].shape))
        df_temp = (df1[i].loc[:, (df1[i] != df1[i].iloc[0]).any()])
        df_temp[label] = df1[i][label]
        dfAll_no_const.append(df_temp)
#        print(str(i) + str(dfAll_no_const[i].shape))
    return dfAll_no_const
#---------------------------------------------------------------------------
#function - drop columns that do not repeat in other dataframes.
def drop_umatching_columns(dfs, label_column = '', drop_column = ''):
    #create dictionarry of data frame columns
    col_dic = []
    for i in range(len(dfs)):
        col_dic.append(dfs[i].columns.values.tolist())
    print('number of files : ' + str(len(col_dic)))
    a = set.intersection(*[set(list1) for list1 in col_dic])
    ab = sorted(list(a))
    if drop_column != '' : ab.remove(drop_column)
    if label_column != '' :
        ab.remove(label_column)
        ab.append(label_column)
#    return ab
    for i in range(len(dfs)):
        dfs[i] = dfs[i][(ab)]
    print('number of same atributes : ' + str(len(ab)))
    return dfs
#---------------------------------------------------------------------------
#function - remove non ascii characters from column names
def remove_non_ascii_column(df_all_la):
    def remove_non_ascii(text):
        return unidecode(unicode(text, encoding = "utf-8"))
    col_2 = []
    col_1 = df_all_la.columns.values.tolist()
    for i in range(len(col_1)):
        a = remove_non_ascii(col_1[i]).replace(' ', '_')
        a = a.replace('%', 'prc')
        col_2.append(a)
#    df_all_new_names = df_all_la
    for i in range(len(df_all_la.columns.values.tolist())):
        df_all_la.rename(columns={col_1[i]:col_2[i]},inplace=True)

    df_all_la = df_all_la.reset_index(drop=True)
#    return df_all_la
#---------------------------------------------------------------------------
#convert columns to numeric, 1 df as input
def convert_numeric(df_input, expetion_columns=''):
    for j in range(len(df_input.columns.values.tolist())-1):
        if df_input.columns[j] == 'Device_Time' or df_input.columns[j] == 'driver':
            continue
        df_input.iloc[:,j] = pd.to_numeric(df_input.iloc[:,j], errors='coerce')
        df_input.fillna(0)
#---------------------------------------------------------------------------
#create csv file
def write_csv(dframe, name='', pathtofolder='', delimiter='\t'):
    #check if path is given
    if pathtofolder == '':
```

```
        path_to_results = '../driving data/colt-all-atributes/concat/'
    else:
        path_to_results = pathtofolder
    if delimiter=='\t':
        delim = 'tab'
    else:
        delim = str(delimiter)
    atributes = str(len(dframe.columns.tolist()))
    #check if name provided
    if name == '' :
        name = atributes + '_atributes'

    ts = time.time()
    st = datetime.datetime.fromtimestamp(ts).strftime('%Y%m%d_%H%M')
    dframe = dframe.fillna(0)
      dframe.to_csv(path_to_results + name +'_'+ delim +'_'+ st + '.csv', sep=delimiter, encoding='utf-8',
index=False, na_rep='')
#------------------------------------------------------------------------
#write pandas DataFrame to .arff file for WEKA.
#arguments passed: (dataFrame, name='', path_to_folfer='/', label_column='')
# dataFrame - comuplsary. dataframe to export
# name - optional. Start name of the file appended with tymestamp. if no name only time stamp
# path_to_folder - optional. If no path specified, saves in current folder.
# label_column - optional. Column containing class labels. Takes last if not provided.
def write_arff(dataFrame, name='', pathtofolder='', label_column=''):
    if pathtofolder == '':
        path_to_results = '../driving data/colt-all-atributes/concat/'

    ts = time.time()
    st = datetime.datetime.fromtimestamp(ts).strftime('%Y%m%d_%H%M')
    filename = path_to_results + str(name) + '_' + st + '.arff'
    # Open the file with writing permission
    myfile = open(filename, 'w')

    #write file heading
    myfile.write('%blabla\n\n')
    myfile.write('@relation obd2\n')

    #meke column names and types
    for c in dataFrame.columns:
        if (dataFrame[c].dtype == 'object'):
            wr = '@attribute ' + str(c) +' {' + (','.join(map(str,dataFrame[c].unique()))) + '}'
            myfile.write( wr + '\n')
        else:
            wr = '@attribute ' + str(c) +' numeric'
            myfile.write( wr + '\n')

    # Write data to file
    myfile.write('\n@data\n')

    for j in dataFrame.values:
        wr = (','.join(map(str,j)))
        myfile.write( wr + '\n')

    # Close the file
    myfile.close()

    print(filename + ' created successfuly')
#------------------------------------------------------------------------
#write function to create sliding windows and lag windows
```

```
def roll_lag(df, window_size, lags=0, y_round=3, atributes_to_ignore=''):

    if df.empty:
        print ('no DF passed to function roll_lag')
        return

    if window_size < 2:
        print('window size should be bigger than 1')
        return

    #create temp variables
    df_sythetic = pd.DataFrame()
    result_mean = {}
    result_lag = {}
    result_mean.clear()
    result_lag.clear()

    for name in df.columns:
        if name in atributes_to_ignore:
            continue
        col_name = str(window_size) + 's_mean_' + name
        result = df[name].rolling(window_size).mean().fillna(0).round(y_round)
        result_mean.update({ col_name : result })
        if lags > 0 :
            for j in range(lags):
                lag_name = (str(j+1) + '_lag_' + name)
                y_lag = (j+1) * (-1)
                lag_result = result.shift(periods=y_lag).fillna(0)
                result_lag.update({ lag_name : lag_result })
    for x in result_mean:
        df_sythetic[x] = result_mean[x]
    if len(result_lag) > 0:
        for x in result_lag:
            df_sythetic[x] = result_lag[x]

    df_new = pd.concat([df_sythetic, df], axis=1, sort=False)
    #sort columns in df
    col_list = df_new.columns.values.tolist()
    col_list.remove('driver')
    col_list = sorted(col_list)
    col_list.append('driver')
    df_new = df_new[(col_list)]
    return df_new
#-----------------------------------------------------------------------
# normalize data function.
def prepr_norm(df, ignore='', l = 'l2', axs=1):
    df_temp = pd.DataFrame()
    col = df.columns.values.tolist()
    if ignore:
        for r in ignore:
            col.remove(r)
    x = df[col].values
    normalized_X = preprocessing.normalize(x,norm=l,axis = axs)
    df_temp = pd.DataFrame(normalized_X, columns=col, index = df.index)
    if ignore:
        for r in ignore:
            df_temp[r] = df[r]

    return df_temp
#-----------------------------------------------------------------------
```

Pielikums nr. 11

U-kontūras salīdzinājums ar citām metodēm.

Dataset (# of class)	Rand index			Number of u-shapelets used
	Extracted Features [33]	u-Shapelets	Time Series ED	
Trace (4)	0.74	**1**	0.75	2
Syn-Control (6)	0.85	**0.94**	0.87	5
Gun Point (2)	0.49	**0.74**	0.49	1
ECG (3)	0.4	**0.7**	*not-defined*	1
Population (2)	0.8	**0.9**	0.5	1
Temperature (2)	0.8	0.9	**1**	1
Income (2)	0.5	0.5	0.5	1

Dataset – datu kopa

(# of clases) – klašu skaits datos

Rand index – random index klašu sadalījuma salīdzināšanas metode (prognozētie ar reāliem).

1 – perfekta atbilstība.

Number of u-shapelets used – izmantotās u-kontūras prognozei.

Extraced Features – Izgūto iezīmju metode

u-Shapelets – U-kontūru metode

Time Series ED – Laika rindu Eklīda distances metode

DOKUMENTĀRĀ LAPA

Maģistra darbs “AUTOVADĪTĀJA IDENTIFIKĀCIJA, IZMANTOJOT OBD2 DATUS” izstrādāts LU Datorikas fakultātē.

Darba teksta galīgā versija izgatavota 2019. gada 19. maijā.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ______________________________

(Autora paraksts un datums)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par p i e m ē r o t u / n e p i e m ē r o t u (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: ____________________________________

(Vadītāja paraksts un datums)

Darbs iesniegts maģistratūras sekretariātā _________________________.

(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: ______________________________.

(Metodiķes paraksts)

Recenzents: pētnieks, Dr. dat., Artis Mednis

(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

__________________________ prot. Nr. _________

(Darba aizstāvēšanas datums)

Komisijas sekretārs: ________________________________

(Sekretāra paraksts)