Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Transporto priemonių parametrizavimo, naudojant pasyvius jutiklius, tyrimas

Baigiamasis magistro projektas

Mantas Ambraziūnas

Projekto autorius

Prof. Dangirutis Navikas

Vadovas

Kaunas, 2023

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Transporto priemonių parametrizavimo, naudojant pasyvius jutiklius, tyrimas

Baigiamasis magistro projektas

Elektronikos inžinerija (6211EX012)

Mantas Ambraziūnas

Projekto autorius

Prof. Dangirutis Navikas

Vadovas

Doc. Mindaugas Knyva

Recenzentas

Kaunas, 2023

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Mantas Ambraziūnas

Transporto priemonių parametrizavimo, naudojant pasyvius jutiklius, tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdamas kitų asmenų autoriaus ar kitų teisių, laikydamasis Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs;

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalintas iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Mantas Ambraziūnas

Patvirtinta elektroniniu būdu

Ambraziūnas, Mantas. Transporto priemonių parametrizavimo, naudojant pasyvius jutiklius, tyrimas. Magistro baigiamasis projektas / vadovas prof. Dangirutis Navikas; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, inžinerijos mokslai.

Reikšminiai žodžiai: klasifikavimas, transporto priemonės, pasyvieji jutikliai.

Kaunas, 2023. 54 p.

Santrauka

Šio darbo tikslas yra sukurti transporto priemonių klasifikavimo algoritmą pasinaudojant iš kelių pasyvių jutiklių surinktais duomenimis. Darbo metu buvo tiriamos skirtingų pasyvių jutiklių panaudojimo galimybės transporto priemonių klasifikavimo tikslams.

Pirmoje dalyje apžvelgiami kitų mokslininkų darbai, naudoti jutikliai ir sistemos. Aprašomi populiariausi transporto priemonių klasifikavimo metodai – naudojant vaizdo kameras ir magnetinius jutiklius. Taip pat aprašomi ir keli mažiau populiarūs metodai naudojant įvairius kitus jutiklius. Galiausiai palyginami rezultatai, kuriuos gavo straipsnių autoriai naudodamiesi aprašytais metodais ir sistemomis.

Antroje dalyje aprašoma projekto duomenų surinkimo sistema. Paaiškinamas jos veikimo principas ir aprašomi parinkti jutikliai bei sistemos parametrai.

Trečioje dalyje pateikiama 4 rūšių jutiklių analizė – magnetinių, akselerometrų, mikrofonų ir pjezoelektrinių gembių. Aprašomi iš šių jutiklių surinkti duomenys ir pateikiama jų analizė. Daugumai klasifikavimo modelių ir metodų reikalingi požymiai, taigi šioje dalyje taip pat aprašomi naudoti požymiai ir metodai, kuriais šie požymiai buvo apskaičiuoti. Galiausiai pateikiamos išvados apie kiekvieno tipo jutiklio suteikiamą informaciją klasifikavimo tikslams.

Ketvirtoje dalyje aprašomi trys skirtingi klasifikavimo modelių tipai. Pateikiami klasifikavimo rezultatai klasifikuojant į 5, 6 ir 7 klases, pasinaudojant įvairiais požymiais, gautais iš surinktų duomenų. Gauti rezultatai yra palyginami ir aprašomos galimos jų panaudojimo situacijos.

Ambraziūnas, Mantas. Investigation of Vehicle Parametrization Using Passive Sensors. Master's Final Degree project / supervisor Prof. Dangirutis Navikas; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): Electronics Engineering, Engineering Sciences.

Keywords: classification, vehicles, passive sensors.

Kaunas, 2023. 54 p.

Summary

The main goal of this work is to create an algorithm for classifying vehicles using data gathered from multiple passive sensors. During the research, various possibilities of using passive sensors for the purpose of vehicle classification were investigated.

The first part reviews the work of other scientists, the sensors and systems used in their research. It describes the most popular methods of vehicle classification using video cameras and magnetic sensors. Several less popular methods using other sensors are also described. Finally, the results obtained by the authors of the articles using the described methods and systems are compared.

The second part describes a data collection system of the project. The principle of its operation is explained, and the sensors and system parameters are described.

The third part presents an analysis of 4 types of sensors - magnetic, accelerometers, microphones and piezoelectric sensors. Data collected from these sensors are described and analyzed. Most of the classification models and methods require extracting features from collected data therefore, this section also describes the methods used and the methods that need to be calculated. Finally, conclusions are drawn on the information provided by each type of sensor for classification purposes.

The fourth part describes three different types of classification models. Classification results are presented for classification into 5, 6 and 7 classes using different methods obtained from the collected data. The obtained results are compared, and possible situations of their use are described.

Turinys

Santrumpų ir terminų sąrašas

Santrumpos:

TP – transporto priemonė;

ŽDF – žemų dažnių filtras;

DI – dirbtinis intelektas;

Terminai:

TCP – tinklo ryšio protokolas, skirtas duomenų paketams siųsti internetu (angl. Transmission Control Protocol);

MEMS – (angl. *Micro-electro-mechanical*) mažas mechaninis modulis, varomas elektros energijos;

Fluxgate – magnetinis jutiklis, kuris magnetinio lauko matavimui naudoja rites;

Klaidų matrica – lentelė, kurioje atvaizduojamos tikimybės kiekvieną klasę priskirti vienai iš galimų klasių pasinaudojant klasifikavimo modeliu. Idealiu atveju įstrižainėje yra tikimybės lygios vienetui, o visur kitur – nuliui;

LASSO – (angl. *least absolute shrinkage and selection operator*) reguliatorius, naudojamas DI modeliuose siekiant išvengti permokymo reiškinio ir naudingų požymių atrinkimui;

Įvadas

Transporto priemonių identifikavimas ir klasifikavimas yra svarbi išmanaus miesto dalis. Šiai užduočiai atlikti yra pasinaudojama įvairių jutiklių suteikiamais duomenimis. Klasifikuojant transporto priemones galima nukreipti transporto srautus tinkama kryptimi tam, kad būtų sumažintos spūstys, taip pat autoįvykių kiekis ir oro tarša. Transporto priemonių klasifikavimo sistemų diegimas yra vienas galimų sprendimų kuris padėtų pasiekti Europos sąjungos užsibrėžtą tikslą iki 2030 metų sumažinti oro taršą 55 % lyginant su 1990 metais. Identifikuojant transporto priemones taip pat galima greičiau surinkti kelio ar stovėjimo mokesčius, be to, fiksuojant transporto priemonių greitį, taip pat yra padidinamas vairuotojų atidumas greičio ribojimams tai kelio atkarpai. Visa tai įvertinus galima teigti, jog jutiklių panaudojimas keliuose yra naudingas kalbant apie saugumą ir ekonomiškumą keliuose bei ekologinį aspektą siekiant sumažinti oro taršą.

Darbo tikslas - sukurti transporto priemonių klasifikavimo algoritmą pasinaudojant iš kelių pasyvių jutiklių surinktais duomenimis.

Darbo uždaviniai:

1. atlikti jau egzistuojančių transporto priemonių klasifikavimo metodų ir jiems naudojamų jutiklių analizę;
2. paruošti surinktus duomenis tyrimui juos suskirstant pagal atitinkamas transporto priemonių klases;
3. ištirti skirtingų pasyvių jutiklių panaudojimo galimybes transporto priemonių klasifikavimui;
4. sukurti transporto priemonių klasifikavimo algoritmą pasinaudojant kelių pasyvių jutiklių duomenimis.

1. Transporto priemonių klasifikavimo metodų apžvalga

Pastaraisiais metais daugumoje mokslinių tyrimų, susijusių su TP klasifikavimu buvo naudojami magnetiniai jutikliai ir vaizdo kameros, bet pasitaiko ir įvairesnių duomenų nuskaitymo metodų, tokių kaip radijo švyturėliai, šviesolaidžiai, mikrofonai ir infraraudonųjų spindulių bei ultragarso jutikliai. Klasifikavimo algoritmai būna paremti dirbtiniu intelektu ir / arba požymių atrinkimu. Toliau šiame skyriuje aprašomi kitų mokslininkų transporto priemonių klasifikavimo tyrimai.

1.1. Vaizdo kameros

Labiausiai skirtingas TP klases skiriantys požymiai yra išvaizda, kėbulo forma, dydis – visi šie požymiai gali būti išskirti pasinaudojant vaizdo kamera. Mokslininkų tyrimuose, kuriuose TP klasifikavimui buvo naudojamos vaizdo kameros, buvo pasiekti vieni geriausių klasifikavimo tikslumų lyginant su kitais metodais, taip pat klasifikuojama buvo į gana daug skirtingų klasių, kai kuriais atvejais buvo atpažįstami net TP modeliai. Pavyzdžiui, M. Biglari pasinaudojo tuo metu jau sudaryta duomenų baze, kuri buvo skirta klasifikavimo iššūkiui ir išskyrė 281 TP klasę su 95 % tikslumu (žr. 1 pav.) [1].

1 pav. Klasifikavimo, naudojant TP nuotraukas, rezultatai [1]

Kaip galima pastebėti, transporto priemonės buvo atpažintos net visiškoje tamsoje, nors yra nustatyta, kad suprastėjęs klasifikavimo tikslumas esant prastam ar besikeičiančiam apšvietimui yra viena iš klasifikavimo, naudojant vaizdo kameras, problemų [2]. Kiti veiksniai sukeliantys sunkumų šiems metodams yra vaizdo uždengimas, pavyzdžiui, dėl rūko ar lietaus. TP klasifikavimas naudojant vaizdo kameras reikalauja didesnių skaičiavimo resursų ir daugiau energijos, lyginant su kitais metodais. Taip yra dėl to, kad nuotraukos ar vaizdo įrašai, priklausomai nuo rezoliucijos, užima didelį atminties kiekį, be to, ne visa informacija viename kadre būna naudinga, pavyzdžiui, visais atvejais fonas (kelias, medžiai, pastatai ir t.t.) yra nenaudingas, bet tam, kad kadre būtų galima aptikti TP, reikia išanalizuoti visus jame esančius pikselius.

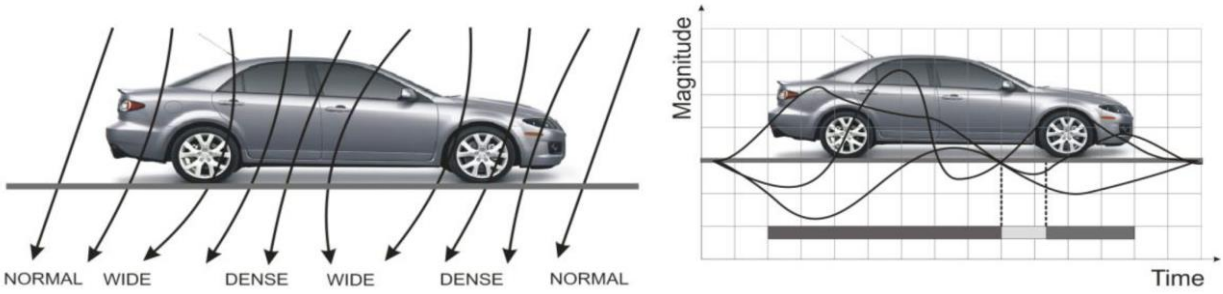
Naudojant ne aukštos kokybės nuotraukas būna klasifikuojama nebe pagal TP modelius, bet pagal jų tipus, pavyzdžiui, motociklai, furgonai, autobusai ir t.t. Naudodami eismo stebėjimo kamerų nuotraukas, M. Biglari su komanda suklasifikavo TP į 21 klasę su 97 % tikslumu [1], o M. A. Hedeya straipsnyje aprašė metodą, kuriuo galima atlikti klasifikavimą į 10 klasių su 97,6 % tikslumu [2].

Vietoje nuotraukų klasifikavimui gali būti naudojamo vaizdo įrašai. Šiuo metodu gaunami kiek prastesni rezultatai negu atliekant klasifikavimo uždavinius su nuotraukomis, tačiau paminėtina tai, kad metodas taip pat turi ir pranašumų, vienas jų – šešėlių panaikinimas. Šešėliai gali būti aptikti kaip dalis TP, taip iškraipant aptikto objekto formą ir dėl to ji gali būti priskirta neteisingai klasei. Vaizdo įrašą sudaro daug nuotraukų, taigi, lyginant kiekvieną kadrą su ankstesnių, galima tiksliau nustatyti transporto priemonės kontūrus – nejudantis fonas yra tiesiog panaikinamas ir tada lieka tik judantys objektai, kurių vienas ir yra analizuojamoji transporto priemonė [3] [4]. Šio apdorojimo metu kartu yra iš dalies panaikinamas ir šešėlis dėl to, kad nors jis yra ir judantis, jis taip pat turi fono kontūrus, tai yra, po juo matosi kelio nelygumai, eismo juostų linijos ir panašiai. Vykdant TP klasifikavimą pagal vaizdo įrašus taip pat galima stebėti ir transporto priemonių judėjimo greitį – didesnės ir sunkesnės transporto priemonės juda lėčiau, pradeda anksčiau stabdyti, yra ne tokios manevringos. Vis dėlto, naudojant šį klasifikavimo metodą kyla sunkumų dėl triukšmo, atsirandančio iš įvairių kitų judančių objektų – medžių lapai, debesų šešėliai ir panašiai [3]. Y. Chen pavyko suklasifikuoti TP į 2 klases (lengvos ir sunkios TP) su 95 % tikslumu.

Kitas klasifikavimo metodo pranašumas, naudojant vaizdo kameras, yra tas, kad vaizdo kameros yra įdiegtos įvairiose miesto vietose kitoms reikmėms, pavyzdžiui, miestų sankryžų. Šias vaizdo kameras galima panaudoti tiesiogiai tokiems metodams kurti ir testuoti. Kadangi dažniausiai šio tipo klasifikavimo metodai naudoja dirbtinį intelektą, vienas reikalavimų geram modeliui sukurti yra didelis duomenų kiekis – kuo daugiau duomenų – tuo geriau, o turint daug miestuose įdiegtų vaizdo kamerų duomenų kiekis nėra labai didelė problema. Taip pat internete yra sukauptų didelių duomenų bazių (pavyzdžiui, kaip aprašoma [5]), kurioms būna sukuriami specialūs iššūkiai skirti įvairioms problemoms spręsti, įskaitant ir transporto priemonių klasifikavimą.

1.2. Magnetiniai jutikliai

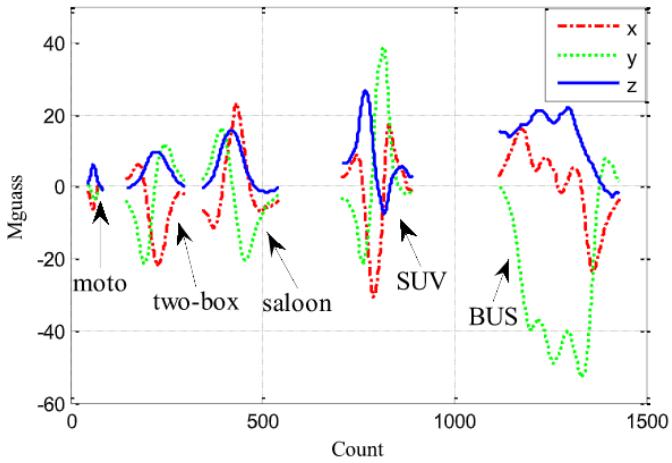
Transporto priemonių kėbulas visada būna pagamintas iš metalo, be to, metalo yra ir kitose dalyse, todėl transporto priemonės turį įtakos aplinkiniam magnetiniam laukui (žr. 2 pav.).



2 pav. Transporto priemonės iškraipomas magnetinis laukas (kairėje) ir magnetinis laukas, išmatuotas 3 ašių magnetiniu jutikliu (dešinėje) [6]

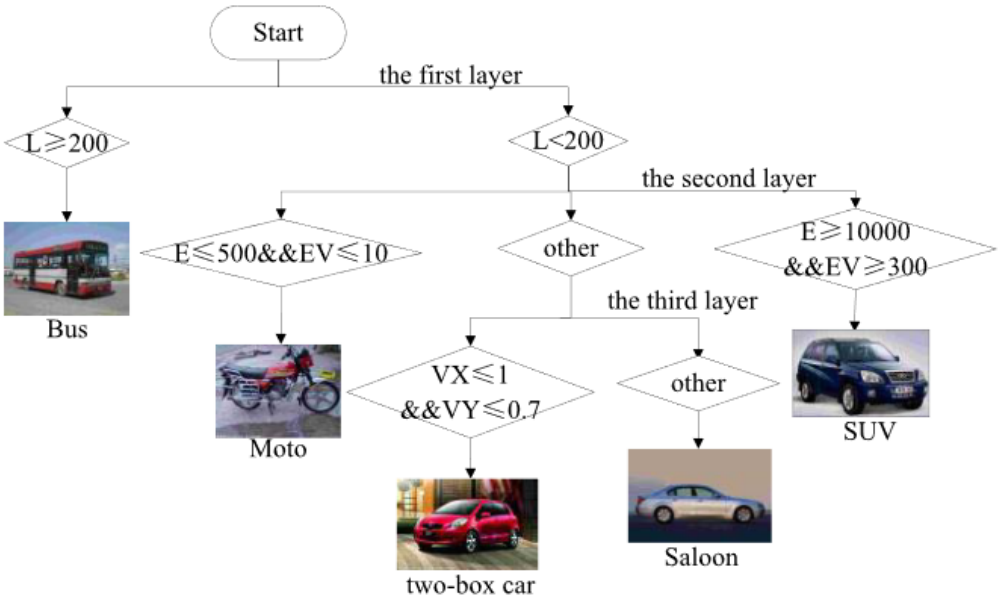
Šiuo principu paremti metodai, kurie siekia transporto priemones stebėti ir klasifikuoti pasinaudojant magnetinio lauko jutikliais. Vienas seniausių būdų aptikti transporto priemones yra magnetinės kilpos – jos yra tinkamos ir TP aptikti, ir skaičiuoti. Vis dėlto klasifikavimo užduočiai atlikti dažniausiai yra naudojami MEMS arba *fluxgate* tipų magnetiniai jutikliai dėl didesnio tikslumo, mažesnės galios ir jutiklio dydžio.

TP klasifikavimo uždavinys, pasinaudojant magnetinių jutiklių duomenis, yra sprendžiamas dviem būdais – pasinaudojant dirbtinį intelektą ir jo nenaudojant. Nenaudojant dirbtinio intelekto yra sudaromas sprendimų medis, kuriame surašomi įvairūs parametrai, išgauti iš surinktos magnetinių signalų duomenų bazės. Tai yra įmanoma dėl to, kad skirtingo tipo TP skirtingai iškraipo magnetinius laukus (žr. 3 pav.).



3 pav. Skirtingoms TP važiuojant gautos magnetinės signatūros [7]

Kaip galima matyti 3 paveiksle, kuo didesnė TP, tuo didesnė magnetinių iškraipymų amplitudė ir tuo daugiau taškų joje yra, be to, paminėtina tai, kad skiriasi ir signatūros forma. Norint priskirti naują transporto priemonę kuriai nors klasei, magnetiniai signalai yra išanalizuojami ir iš jų išrenkami įvairūs parametrai ir galiausiai pagal sudarytą sprendimų medį priimamas sprendimas. Tokio medžio pavyzdys pateiktas toliau (žr. 4 pav.).

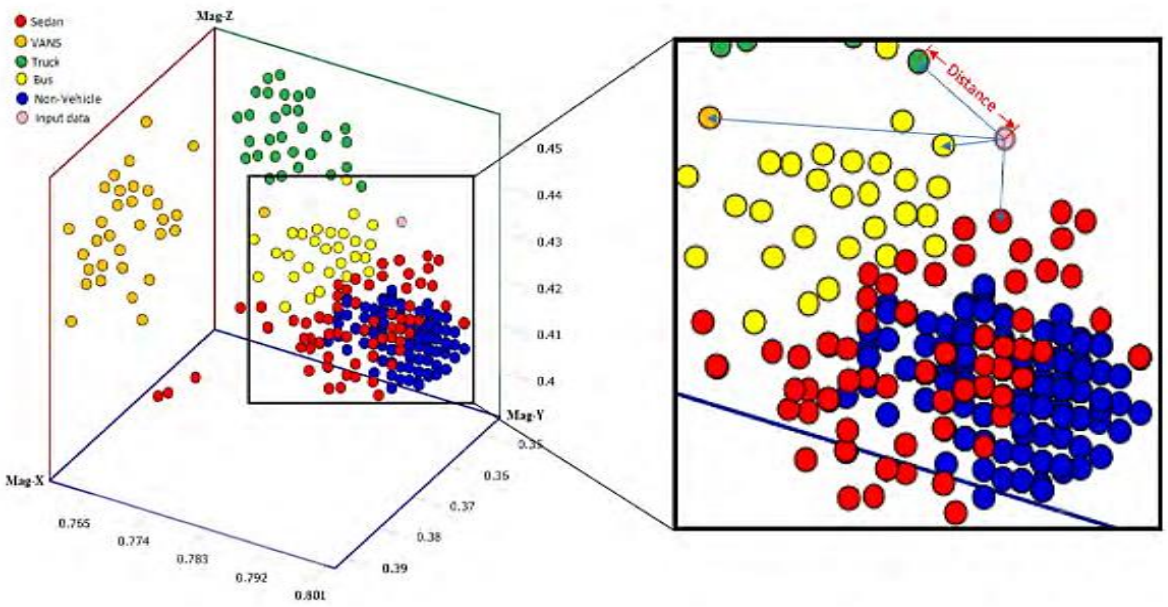


4 pav. Sprendimų medis, skirtas klasifikuoti į 4 klases [7]

Išskiriami parametrai būna tokie kaip signalo ilgis, maksimali amplitudė, išvestinės nulio kirtimų kiekis, TP greitis, signalo dalių didėjimo ir mažėjimo greitis ir panašiai. Greitis yra gana svarbus

parametras klasifikavimui, bet tam reikalingi du jutikliai pastatyti žinomu atstumu vienas nuo kito – pravažiavus transporto priemonei vykdoma koreliacija tarp dviejų gautų signatūrų tam, kad būtų nustatytas laiko skirtumas tarp jų, kuris panaudojamas suskaičiuoti greičiui. Greitis yra panaudojamas apskaičiuoti TP ilgiui, kuris yra tinkamas parametras klasifikavimui, nes skirtingų tipų transporto priemonių ilgiai įprastai skiriasi.

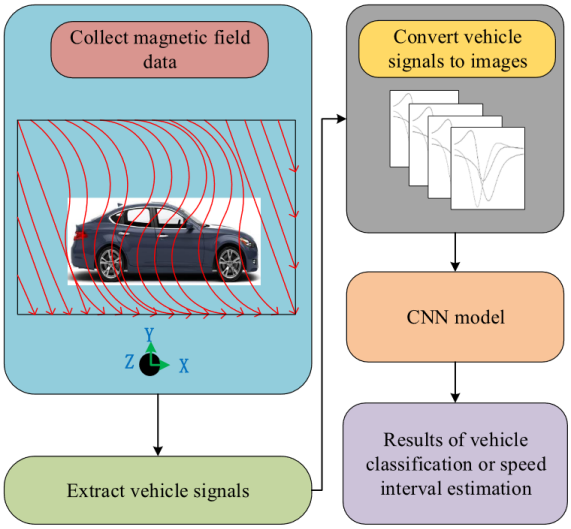
Kaip alternatyva klasifikavimui pagal sprendimų medį, naudojamas dirbtinis intelektas. Požymių atrinkimas vis tiek yra reikalingas, bet šiuo atveju sprendimus priima dirbtinio intelekto modelis. Pavyzdžiui, X. Chen iš atrinktų parametrų suformavo vieną vektorių ir jį panaudojo klasifikavimui (žr. 5 pav.) [8].



5 pav. Skirtingų klasių pasiskirstymas 3D erdvėje [8]

5 paveiksle pavaizduota kaip 3D erdvėje išsidėsto apskaičiuoti skirtingų TP vektoriai, išgauti iš TP magnetinio lauko iškraipymų signalo. Norint priskirti naują transporto priemonę kuriai nors klasei patikrinama kuriam taškų debesiui naujas taškas yra artimiausias. Toliau autoriai šiuos duomenis panaudojo dirbtinio intelekto modeliui apmokyti ir galiausiai pasiektas klasifikavimo į 5 klases tikslumas buvo didesnis nei 93 %.

Dar viena alternatyva klasifikavimui pagal magnetinių signalų duomenis yra konvoliuciniai neuroniniai tinklai. W. Li ir jo komanda siekė nustatyti transporto priemonių klasę ir greitį pasinaudodami vienu 3 ašių jutikliu, principas pavaizduotas 6 paveiksle [9].



6 pav. Klasifikavimo ir greičio nustatymo pagal vieno magnetinio jutiklio duomenis procesas [9]

Nuskaitytas magnetinio jutiklio signalas būdavo išsaugomas kaip paveiksliukas, o tada pagal surinktus paveiksliukus būdavo apmokomas modelis. Šiuo atveju požymių atrinkimas nebuvo reikalingas dėl to, kad klasifikavimui tiesiogiai buvo panaudojamas magnetinio lauko iškraipymų signalas. Tam, kad tyrimo metu rezultatus būtų galima patikrinti, buvo naudojama vaizdo kamera (tam, kad būtų žinoma, kokiai TP priklauso kiekvienas signalas) ir greičio radaras (tam, kad būtų žinomas tikras TP greitis). Autoriams pavyko suklasifikuoti transporto priemones į 7 klases su ~98 % tikslumu, bei nustatyti jų greitį su ~97 % tikslumu. [9]

1.3. Kitos TP klasifikavimo sistemos ir metodai

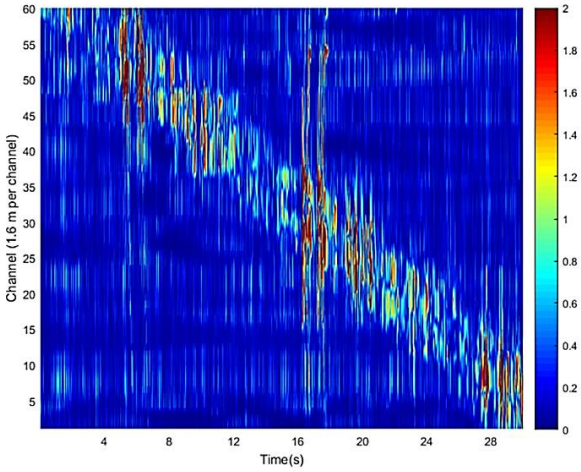
Šiame skyriuje aprašomi mažiau populiarūs metodai ir sistemos negu vaizdo kameros ir magnetiniai jutikliai.

- **Šviesolaidžiai**

Šis duomenų surinkimo metodas pasinaudoja šviesolaidžių šviesos perdavimo savybėmis. H. Liu su komanda įdiegė 320 m šviesolaidį kelio pakraštyje ir pasinaudojo TP sukeliamų vibracijų matavimui (žr. 7 pav.) [10].

7 pav. Kelio krašte įdiegtas šviesolaidis [10]

Šviesolaidžiu periodiškai perduodami lazerio impulsai, kurie pasiekę šviesolaidžio pabaigą yra atspindimi atgal. Važiuojant TP šalia šviesolaidžio, jame būna sukeliamos vibracijos, kurios sutrikdo šviesos sklidimą ir dėl šių trikdžių dalis impulso grįžta anksčiau. Pagal išmatuotą sklidimo laiką apskaičiuojama vieta, kurioje yra transporto priemonė. Sekant vietos kitimą laike galima apskaičiuoti ir TP judėjimo greitį. 8 paveiksle pavaizduoti surinkti duomenys TP važiuojant per stebimą zoną.

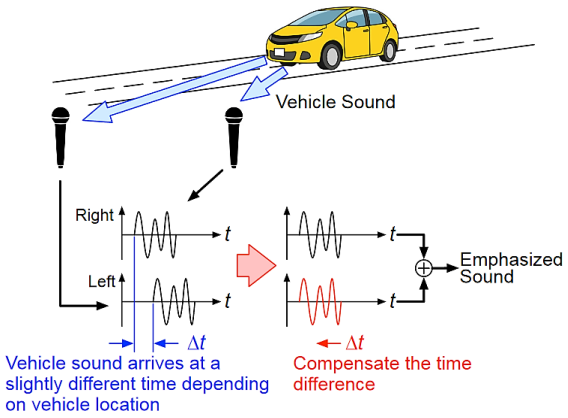


8 pav. Surinkti duomenys važiuojant transporto priemonei [10]

Šio straipsnio autoriai klasifikavimą vykdė pasinaudodami dirbtiniu intelektu. Sistema geriausiai atpažindavo sunkvežimius dėl jų ilgio, masės ir ašių skaičiaus, bet net ir jų klasifikavimo tikslumas siekė tik ~78 %, o bendras sistemos klasifikavimo tikslumas į 3 klases gautas ~72 %. [10]

- **Mikrofonai**

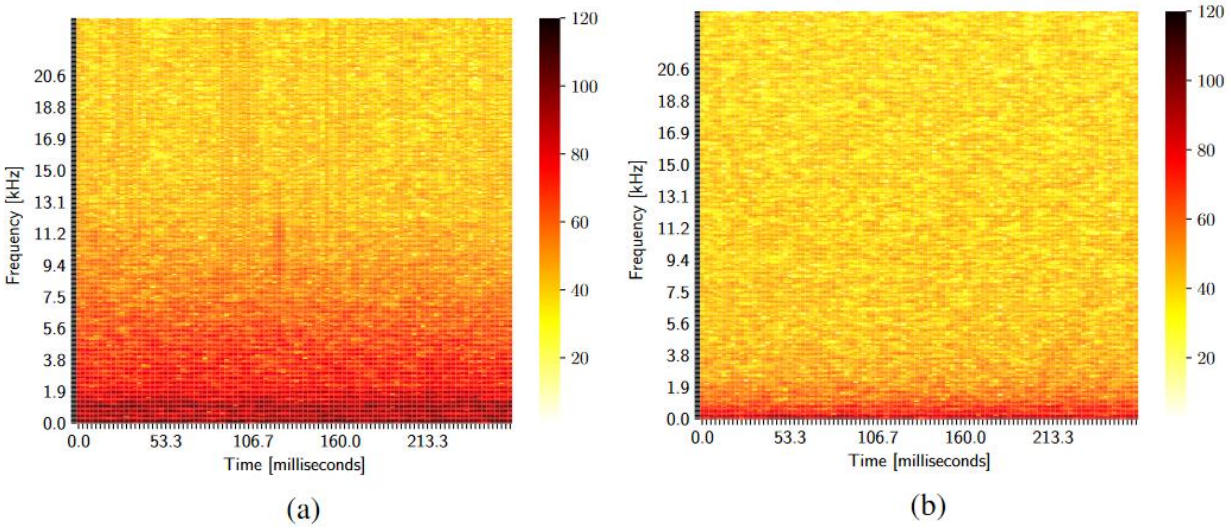
Skirtingų tipų TP skleidžia skirtingus garsus, pavyzdžiui, lyginant sunkvežimio ir lengvojo automobilio skleidžiamus garsus, sunkvežimio skleidžiamas garsas yra duslesnis, turi daugiau žemo dažnio dedamųjų. B. Dawton aprašė eksperimentą, kurio tikslas buvo klasifikuoti TP panaudojant duomenimis iš 2 mikrofonų [11]. Lyginant vieno ir dviejų mikrofonų sistemas, pastarosios turi krypties nustatymo pranašumą – atliekant koreliaciją signalui galima nustatyti transporto priemonės judėjimo kryptį ir greitį, be to, paslinkus signalus taip, kad jie sutaptų ir juos sudėjus gaunamas sustiprintas signalas (žr. 9 pav.).



9 pav. Signalų skirtumas dėl garso sklidimo greičio [11]

Antrasis mikrofonas taip pat duoda informacijos apie kitas tuo pačiu metu važiuojančias TP. Sustiprintame signale atsiradusios stiprios naujos dažninės dedamosios parodo, kad yra kita TP, kuri važiuoja priešinga kryptimi nei pirmoji, o atsiradusios nedidelės dažninės dedamosios – apie ta pačia kryptimi važiuojančią arba visai nevažiuojančią TP. [11]

Klasifikavimas pagal šį metodą yra atliekamas stebint signalus dažnio – laiko srityje. 10 paveiksle pavaizduotos dvi spektrogramos, kai nevažiavo jokios transporto priemonės ir pravažiavus vienai TP.



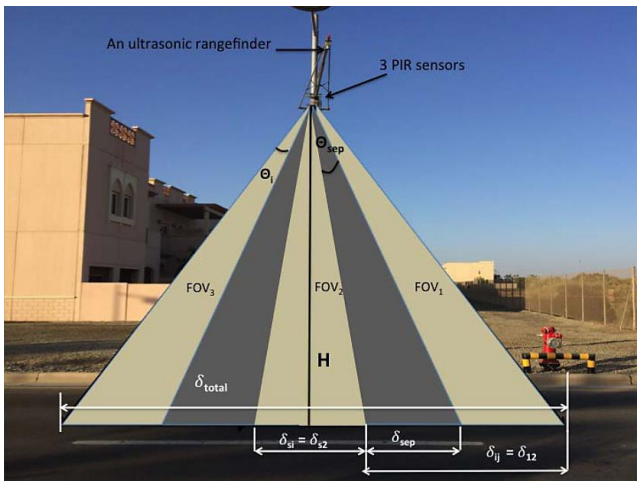
10 pav. Spektrogramos, kai pravažiavo (a) automobilis ir (b) kai nevažiavo jokia transporto priemonė

10 paveiksle galima pastebėti, kad daugiausia informacijos yra sukaupta žemuosiuose dažniuose iki 10 kHz. Prieš klasifikavimą signalas būdavo filtruojamas žemų dažnių filtru tam, kad sumažinti aplinkos triukšmą. Klasifikavimas buvo atliekamas į 3 klases – lengvieji automobiliai, dviračiai ir autobusai. Pasiektas bendras klasifikavimo tikslumas – 95 %, geriausiai sistema atpažindavo autobusus.

Eksperimento metu taip pat buvo palyginti rezultatai su vienu ir su dviem mikrofonais. Neskaitant anksčiau paminėtų pranašumų naudojant du mikrofonus, klasifikavimo tikslumas su 2 mikrofonais buvo 5 % didesnis.

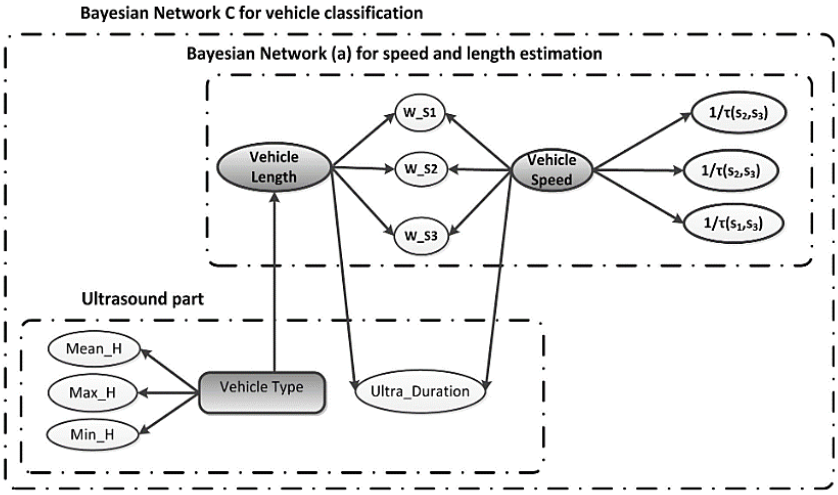
- **Infraraudonųjų spindulių ir ultragarsiniai jutikliai**

Pasyvūs infraraudonųjų spindulių (PIR) ir ultragarsiniai jutikliai yra dar vieni mažos galios ir kainos jutikliai, kuriuos taip pat galima panaudoti greičio matavimui ir TP klasifikavimui. E. Odat su komanda sukūrė ir aprašė sistemą, kurioje integravo šių dviejų tipų jutiklius [12]. Įdiegta sistema pavaizduota 11 paveiksle.



11 pav. Įdiegta sistema su 3 PIR ir 1 ultragarsiniu jutikliais [12]

Anot autorių, atskirai šių jutiklių duomenys nėra informatyvūs dėl mažo diskretizavimo dažnio, tikslumo ir didelio triukšmo, bet sujungus kelių jutiklių duomenis galima gauti norimą rezultatą. Tyrimo metu buvo naudojami Bajeso tinklai jutiklių duomenų sujungimui. Buvo panaudoti du tokie tinklai, kurių vienas turėjo apskaičiuoti transporto priemonės ilgį ir greitį pagal jutiklių matavimus, o kitas – klasifikuoti. Sistemos grafinis modelis pateiktas 12 paveiksle.



12 pav. Sistemos su dvejais Bajeso tinklais grafinis modelis [12]

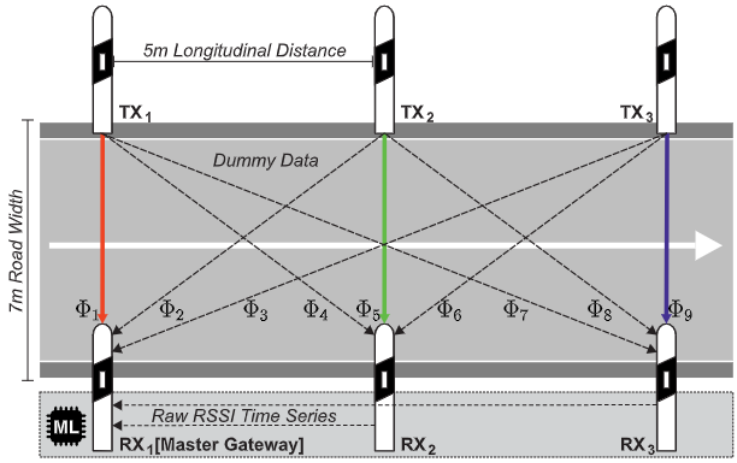
Bajeso tinklams buvo reikalingi požymiai iš surinktų signalų, tokie kaip atstumo matavimo ultragarsiniu jutikliu rezultato vidurkis, maksimali ir minimali vertė bei kiekvieno iš PIR jutiklių nustatytas signalo plotis ir pagal koreliaciją apskaičiuotas greitis. Transporto priemonių aptikimo tikslumas gautas 99 %, vidutinė greičio matavimo paklaida gauta 5 km/h, ilgis vidutiniškai nustatytas su 0,7 m paklaida, o apie klasifikavimo tikslumą neužsiminta.

- **Radijo bangų metodas**

Transporto priemonėse yra gausu metalinių dalių, taigi, jos slopina radijo bangas. Skirtingos TP yra skirtingų formų, taigi jos skirtingai slopina radijo signalą. Šiuo principu remiasi sistemos, siekiančios

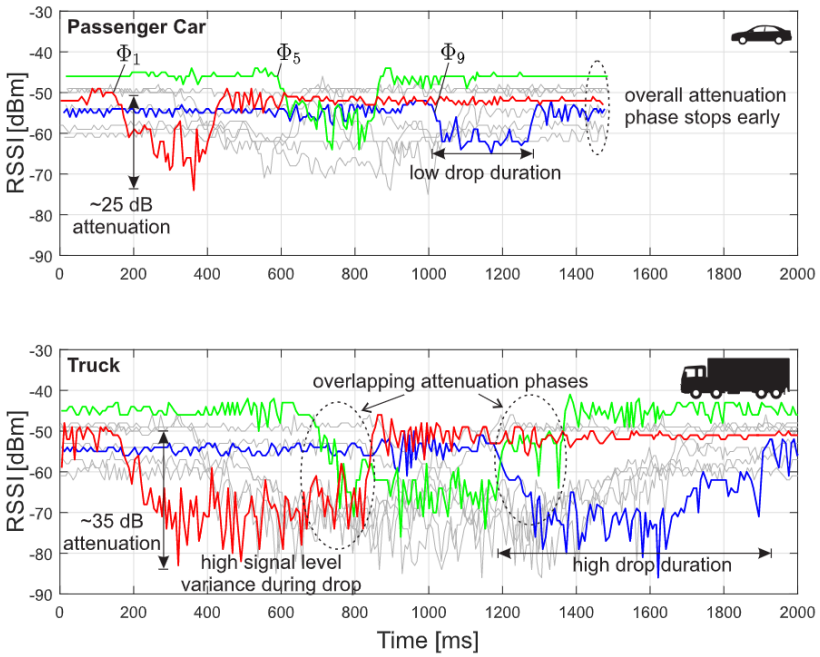
klasifikuoti TP pasinaudojant radijo bangomis. Tokios sistemos dažniausiai naudoja 2,4 GHz dažnių juostos technologijas, tokias kaip *Bluetooth* ar *WiFi*, dėl to jos yra atsparios įvairiems oro reiškiniams, tokiems kaip lietus ir sniegas [13]. Šis dažnių ruožas priklauso nelicencijuotai juostai, taigi tokios sistemos gali veikti tik tose vietose, kur veikia mažai šiuos dažnius naudojančių sistemų, nes jos galėtų sukelti trikdžius. Taigi, tokia sistema negalėtų būti įdiegta mieste dėl pastatuose veikiančio *WiFi* ryšio ar praeivių mobiliuosiuose įrenginiuose veikiančio *Bluetooth* ryšio.

B. Sliwa ir bendraautoriai sukūrė ir aprašė sistemą, kurioje panaudojo 6 radijo ryšio modulius, kurių trys buvo siuntimo režime, o kiti – priėmimo (žr. 13 pav.). [13]



13 pav. Sistemos išdėstymas tyrimui [13]

Sistema buvo įdiegta vienos juostos kelyje, taigi vienu metu galėjo važiuoti tik viena TP. Siųstuvai transliuodavo žinutes imtuvams kas 8 milisekundes ir pravažiuojant TP buvo stebimas priimamo signalo stiprumas. Gauti signalai pavaizduoti 14 paveiksle.



14 pav. Radijo parašai, gauti pravažiavus lengvajai (viršuje) ir sunkiajai (apačioje) TP [13]

Galima pastebėti, kad gauti signalai skiriasi ir slopinimu, ir trukme, ir forma. Gautų laikinių signalų rinkinys vadinamas radijo parašu. Prieš klasifikavimą signalai buvo apdorojami, iš jų apskaičiuojami įvairūs požymiai, tokie kaip greitis, TP ilgis bei važiavimo kryptis. Skaičiavimų ir apdorojimų rezultatai buvo perduodami dirbtinio intelekto modeliui, kuris pagal duomenis nustatydavo TP klasę. Buvo išbandyti keturi modeliai, iš kurių geriausi rezultatai buvo naudojant SVM (angl. *Support Vector Machine*) modelį. Pasiekti klasifikavimo rezultatai pateikti 1 lentelėje.

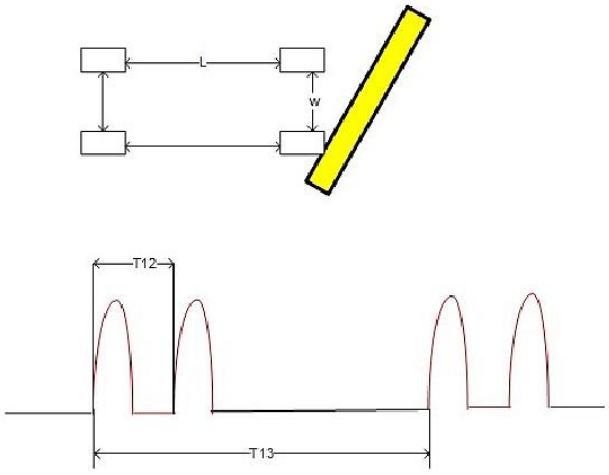
1 lentelė. Radijo švyturėlių metodo klasifikavimo tikslumas

Klasių kiekis	Tikslumas, %
2	99
3	96
7	94

Didžiausias klasifikavimo tikslumas buvo klasifikuojant į dvi klases, bet padidinus klasių kiekį iki 7 klasifikavimo tikslumas nukrito tik per 5 %.

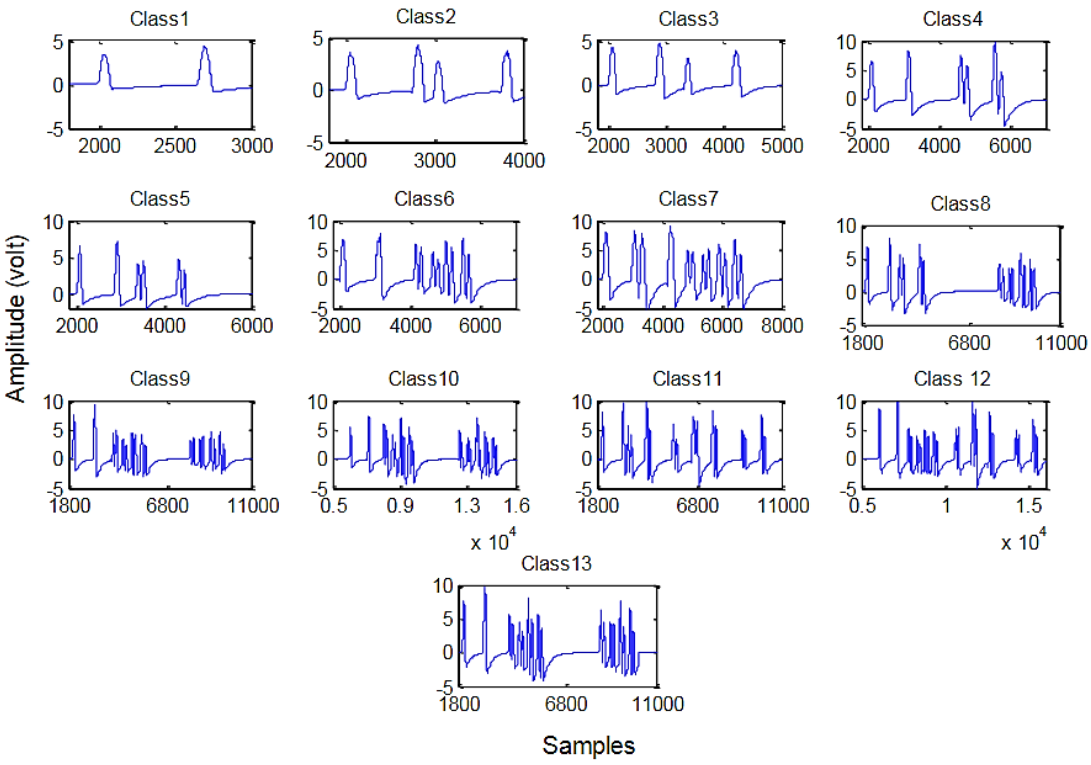
- **Pjezoelektriniai jutikliai**

S. A. Rajab su kolega aprašė transporto priemonių klasifikavimo metodą, kuriame yra panaudojamas vienas pjezoelektrinis elementas [14]. Tyrimo metu elementas buvo pastatytas vienoje eismo juostoje ir pakreiptas 45 laipsnių kampu, kaip pavaizduota 15 paveiksle.



15 pav. Pjezoelektrinio elemento pozicija (viršuje geltonas) ir pravažiuojančios TP sugeneruotas signalas [14]

Transporto priemonei važiuojant per pjezoelektrinį elementą yra sugeneruojamas elektrinis signalas dėl tuo metu jį veikiančių jėgų. Dėl to, kad buvo naudojamas tik vienas elementas, nebuvo galimybės nustatyti TP plotį, todėl skaičiavimams buvo naudojamas vidutinis visų TP plotis. Bandymo tikslas buvo suklasifikuoti transporto priemones į 13 klasių, kurių pagrindinis skirtumas – ratų kiekis. Klasės surūšiuotos nuo mažiausiai ratų turinčios ir lengviausios TP (motociklas) iki daugiausiai ratų turinčios ir sunkiausios TP (fūra su keliomis priekabomis). Matavimo rezultatai pravažiavus visų klasių TP pateikti žemiau (žr. 16 pav.).



16 pav. Pjezoelektrinio elemento sugeneruotas signalas pravažiavus visų klasių TP [14]

Kaip galima pastebėti 16 paveiksle, kuo daugiau ratų turi TP, tuo daugiau signale būna iškilimų, taip pat, kuo sunkesnė TP, tuo didesnė iškilimų amplitudė. Bandymo rezultatai siekė 98,9 % klasifikavimo tikslumą, o motociklų klasifikavimo tikslumas buvo 100 %. Nors klasių buvo 13, jos buvo suskirstytos tiktais pagal ratų kiekį, taigi sistema negalėtų atskirti lengvo sunkvežimio su 2 ašimis nuo įprasto lengvojo automobilio. Vis dėlto, autoriai teigia, kad idealus motociklų klasifikavimas yra labai geras pasiekimas dėl to, kad kitos klasifikavimo sistemos turi problemų atpažįstant šią klasę.

1.4. Metodų apibendrinimas

Iš apžvelgtų sistemų ir metodų, į daugiausiai klasių klasifikuoja ir didžiausią tikslumą turi vaizdo kameras naudojantys metodai. Vis dėlto, mokslininkai ieško ir kitų alternatyvių metodų dėl to, kad vaizdo kamerų metodai turi ir silpnųjų vietų – lyjant lietui, esant rūkui ar prastam apšvietimui klasifikavimas naudojant vaizdo kameras gali tapti net visai neįmanomas, taip pat metodas reikalauja daugiau energijos nei alternatyvūs metodai. Visų apžvelgtų metodų apibendrinimas pateiktas 2 lentelėje.

2 lentelė. Apžvelgtų jutiklių ir metodų apibendrinimas

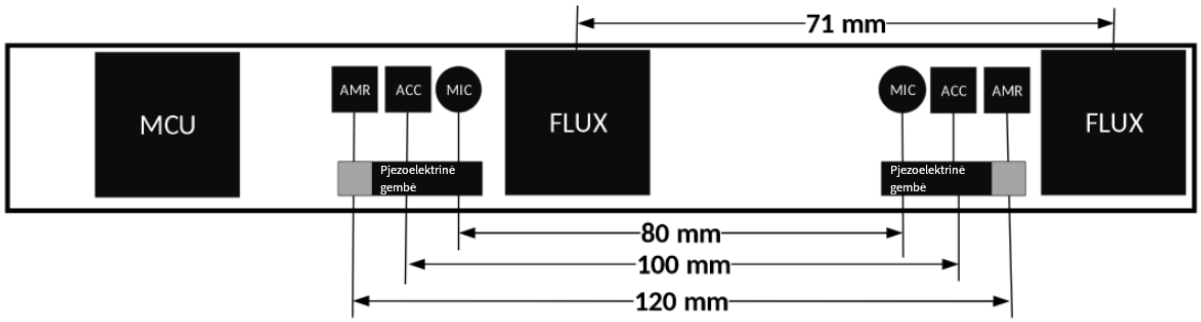
Jutiklis	**Metodas, šaltinis**	**Klasių kiekis**	**Tikslumas, %**	**Mažos galios metodas**	**Jautrus temperatūros pokyčiams**	**Jautrus apšvietimui**	**Jautrus orams**
Vaizdo kamera	Aukštos kokybės nuotraukos [1]	281	95	-	-	+	+
	Žemos kokybės nuotraukos [1]	21	97	-	-	+	+
	Vaizdo įrašai [3]	2	95	-	-	+	+

Jutiklis	**Metodas, šaltinis**	**Klasių kiekis**	**Tikslumas, %**	**Mažos galios metodas**	**Jautrus temperatūros pokyčiams**	**Jautrus apšvietimui**	**Jautrus orams**
Magnetinio lauko	Sprendimų medis [7]	5	94	+	+-	-	-
	Požymių išskyrimas + dirbtinis intelektas [8]	5	93	+	+-	-	-
	Dirbtinis intelektas [9]	7	98	+	+-	-	-
Šviesolaidis	[10]	3	72	+	+-	-	-
Mikrofonai	[11]	3	95	+	-	-	+
Infraraudonųjų ir ultragarsiniai	[12]	-	-	+	-	-	+
Radijo bangų	[13]	2	99	+	-	-	-
		3	96	+	-	-	-
		7	94	+	-	-	-
Pjezoelektriniai	[14]	13	99	+	+-	-	-

Antroje vietoje pagal tikslumą ir klasių kiekį būtų metodas, kuriame naudojami pjezoelektriniai elementai, bet šį metodą nelabai galima lyginti su kitais, nes visuose kituose metoduose nebuvo tokių klasių kaip šiame, t.y. klasės šiame metode yra priskiriamos vien tiktais pagal ratų kiekį, taigi buvo įtrauktos net ir sunkios TP su daug priekabų. Tuo tarpu kituose metoduose net nebuvo svarstoma klasifikuoti TP su priekabomis. Sekantis geriausias metodas – magnetinių jutiklių pasinaudojant dirbtiniu intelektu. Šio metodo autoriams pavyko suklasifikuoti TP į 7 klases su net 98 % tikslumu. Vienintelis šio metodo trūkumas – kintant temperatūrai šiek tiek keičiasi ir magnetinių jutiklių rodomi duomenys, o tai gali turėti įtakos klasifikavimo tikslumui skirtingais metų laikais.

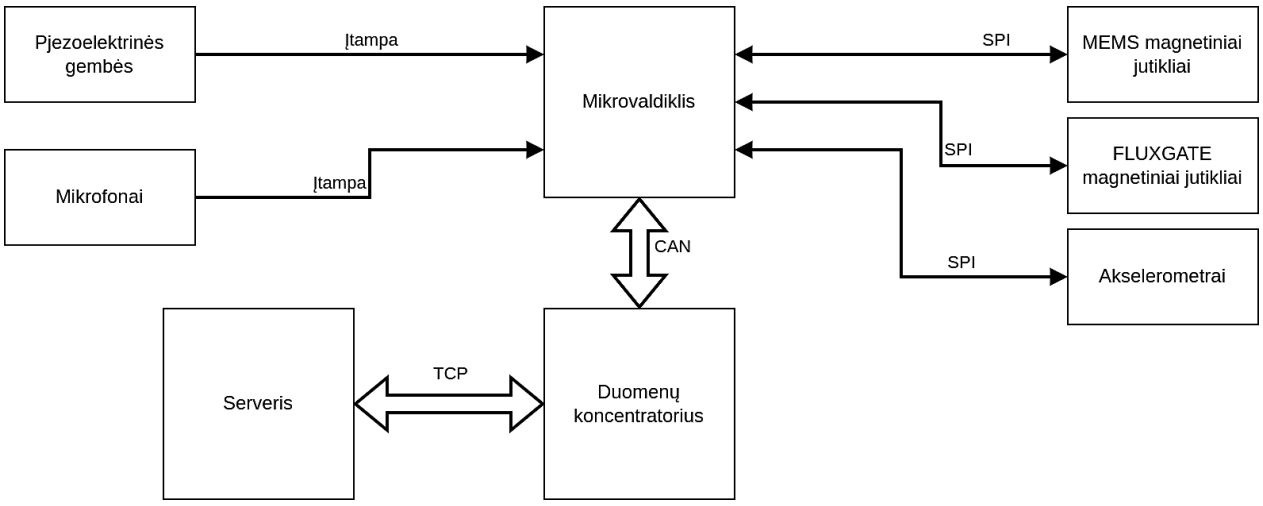
2. Duomenų surinkimo sistema

Duomenų surinkimo sistemai buvo pasirinkta naudoti 5 tipų jutiklius – MEMS ir *fluxgate* tipų magnetinius jutiklius, mikrofonus, akselerometrus ir pjezoelektrines gembes. Magnetiniai jutikliai buvo pasirinkti dėl to, kad jie yra labai populiarūs pasyviniai jutikliai panašaus tipo projektams, kaip aprašyta 1 skyriuje. Jais taip pat yra pasiekiami vieni geriausių klasifikavimo rezultatų, taip pat jais pasinaudojant galima nustatyti TP judėjimo greitį kuris yra svarbus norint apskaičiuoti TP ilgį [7] [8] [9]. Mikrofonai buvo pasirinkti tam, kad būtų galima ištirti ar kelio dangoje integruotais mikrofonais būtų galima gauti panašius rezultatus kaip juos padėjus šalia kelio – nustatyti TP judėjimo kryptį ir greitį, bei atskirti TP pagal jų skleidžiamą garsą kaip aprašė B. Dawton savo straipsnyje [11]. Akselerometrai buvo parinkti tam, kad būtų galima išmatuoti pravažiuojančių TP priemonių sukeliamas vibracijas kelio dangoje. Analogiškas jutiklis akselerometrui – pjezoelektrinės gembės, kurios taip pat gali būti panaudotos vibracijų matavimui. Akselerometrais ir pjezoelektrinėmis gembėmis siekiama nustatyti TP ašių skaičių ir gauti rezultatus, analogiškus aprašytiems [14] straipsnyje. Suprojektuotos sistemos jutiklių dalies (toliau – jutiklių blokas) struktūrinė schema pateikta 17 paveiksle.



17 pav. Jutiklių bloko struktūra

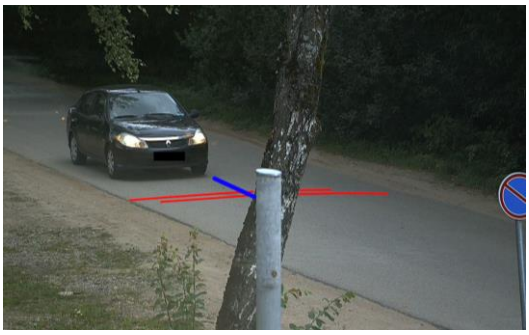
Visų jutiklių buvo naudojama po du tam, kad būtų galima ištirti koreliacijas ar kitokius ryšius tarp jutiklių signalų. Pagal koreliaciją tikimasi pastebėti įvairių rezultatų, tokių kaip mikrofonų atveju krypties nustatymas arba magnetinių jutiklių – greičio įvertinimas. Duomenų surinkimo sistemos struktūrinė schema pateikta 18 paveiksle.



18 pav. Duomenų surinkimo sistemos struktūrinė schema

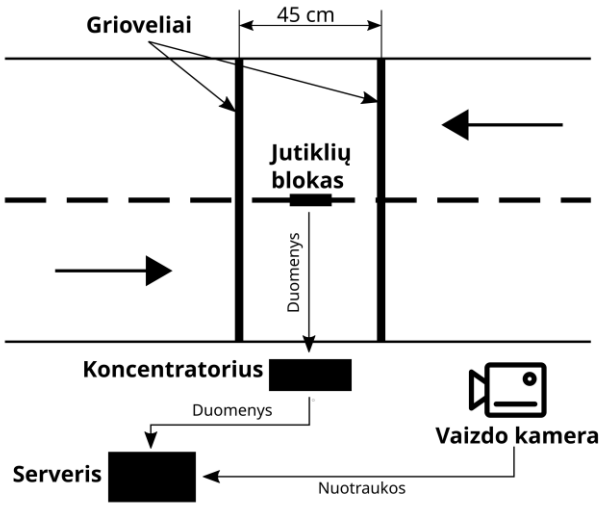
Skaitmeniniai jutikliai (akselerometrai bei MEMS ir *fluxgate* magnetiniai) buvo nuskaitomi per SPI sąsają. Akselerometrų ir MEMS magnetinių jutiklių diskretizavimo dažnis - 1 kHz, o *fluxgate* - 500 Hz. Pjezoelektrinių gembių ir mikrofonų signalai buvo nuskaitomi mikrovaldiklio analogas-kodas keitikliu, atitinkamai 1 kHz ir 4 kHz dažniais. Mikrovaldiklis, surinkęs duomenis iš visų jutiklių juos siųsdavo pasinaudodamas CAN sąsaja duomenų koncentratoriui, kuris duomenis priimdavo ir perduodavo serveriui pasinaudodamas TCP sąsaja. CAN sąsaja buvo naudota dėl tinkamo duomenų perdavimo atstumo ir greičio, be to su šia sąsaja yra patogu prie sistemos prijungti daugiau jutiklių ar net sujungti kelis atskirus jutiklių blokus ir iš jų atskirai surinkti duomenis, pavyzdžiui, kiekviename sankryžos kelyje. TCP sąsaja tarp duomenų koncentratoriaus ir serverio reikalinga tam, kad surinktus duomenis būtų galima perduoti į serverį, kuris galėtų surinkinėti duomenis iš bet kur įdiegtų sistemų. Tokiu atveju pakanka turėti vieną duomenų surinkimo serverį visoms duomenų surinkimo sistemoms.

Jutiklių blokas buvo integruotas dviejų juostų kelio dangoje per vidurį tarp juostų (žr. 19 pav.), tokiu būdu duomenims rinkti iš abiejų eismo juostų pakanka vieno jutiklių bloko. Greičio limitas šioje kelio atkarpoje yra 60 km/h.



19 pav. Kelyje įdiegta sistema. Mėlynai pažymėtas – jutiklių blokas, o raudonai – grioveliai, atsiradę po sistemos integravimo kelio dangoje

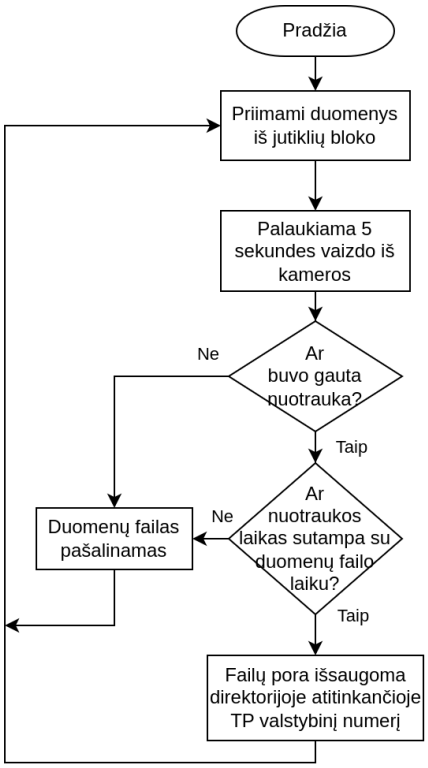
Duomenis iš jutiklių bloko surenka serveris, kuris taip pat priima vaizdus iš šalia kelio pastatytos vaizdo kameros. Vaizdo kamera yra reikalinga patikrinimo tikslams – be jos nebūtų žinoma kokia TP pravažiavo signalų nuskaitymo metu. Pilna duomenų surinkimo sistemos struktūra pateikta 20 paveiksle.



20 pav. Pilna duomenų surinkimo sistemos struktūra

Dėl sistemos diegimo principo, kelyje buvo padaryti du grioveliai, 45 cm atstumu vienas nuo kito. Duomenų koncentratorius buvo pastatytas šalia kelio, o duomenų surinkimo ir saugojimo serveris kartu su kamera – netoliese esančiame pastate.

Duomenys iš jutiklių bloko yra priimami pasinaudojant TCP sąsaja ir išsaugomi kaip failai serverio failų sistemoje. Analogiškai priimamos ir nuotraukos iš vaizdo kameros. Gavus nuotrauką yra vykdomas failų susiejimas pagal jų sukūrimo laikus. Pilnas duomenų surinkimo programos algoritmas pateiktas 21 paveiksle.



21 pav. Duomenų surinkimo sistemos veikimo algoritmas

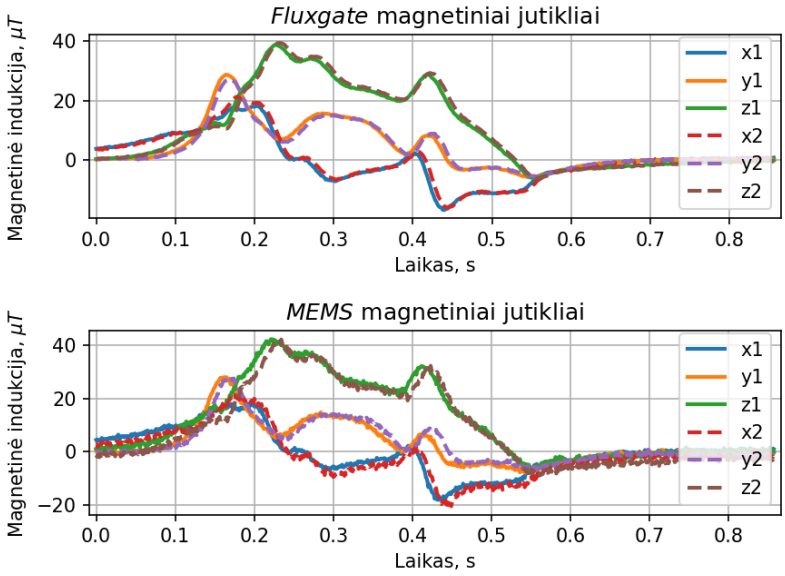
Po jutiklio duomenų priėmimo palaukiama 5 sekundes kol bus gauta nuotrauka iš vaizdo kameros. Tai reikalinga dėl to, kad vaizdo kamera nuotrauką užfiksuoja ir atsiunčia nebūtinai tuo pačiu metu kaip jutiklių blokas – gali atsiųsti prieš TP pravažiuojant arba jai jau pravažiavus. Toliau yra patikrinama ar sutampa nuotraukos ir priimto failo sukūrimo laikai – šis žingsnis reikalingas tiems atvejams, kai kelios transporto priemonės pravažiuoja mažais laiko tarpais. Praėjus 5 sekundėms nuo failo gavimo, jeigu atitinkama nuotrauka nebuvo gauta arba gautų nuotraukų laikai nesutapo su failo laiku – priimtas failas yra pašalinamas, nes per tiek laiko TP jau tikrai bus pravažiavusi ir tai reiškia, kad jos nuotrauka nebuvo užfiksuota, taigi nebebus galima patikrinti kokia transporto priemonė tuo metu važiavo virš jutiklių bloko.

3. Surinktų duomenų analizė

Norint suklasifikuoti TP reikia surinktuose duomenyse surasti požymių, kurie padėtų atskirti vieną TP tipą nuo kito. Šaltiniuose yra aprašyta daug įvairių metodų skiriančiųjų požymių apskaičiavimui ir išgavimui iš signalų – šie požymiai yra tiesiogiai panaudojami klasifikavimo modeliuose. Kaip aprašyta 1 skyriuje, klasifikavimui dažniausiai būna naudojamas vieno tipo jutiklis ir visais atvejais klasifikavimo uždaviniui spręsti yra sudaromas sprendimų medis arba dirbtinio intelekto modelis. Šio darbo tikslas yra sukurti klasifikavimo algoritmą pasinaudojantį kelių jutiklių duomenimis, taigi šiame skyriuje pateikta 4 rūšių jutiklių analizė – magnetinių jutiklių, akselerometrų, pjezoelektrinių gembių ir mikrofonų. Taip pat šiame skyriuje aprašoma kokie metodai buvo naudojami požymių ieškojimui, kokie rasti požymiai gali būti panaudoti klasifikavimui arba ne, taip pat pateikiamos išvados apie kiekvieno iš jutiklių tipų panaudojimą klasifikavimui.

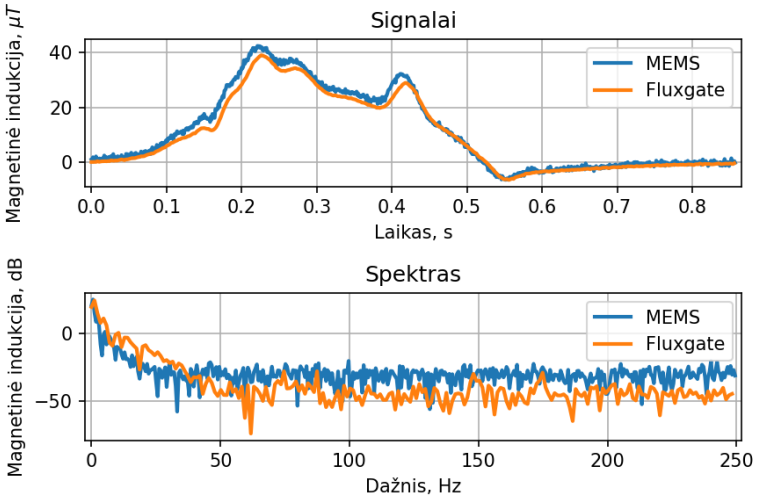
3.1. Magnetiniai jutikliai

Duomenų surinkimo sistemoje buvo naudojami dviejų tipų magnetiniai jutikliai – MEMS ir *fluxgate*. MEMS tipo jutikliai turi mažesnį korpusą ir didesnį diskretizavimo dažnį, o *fluxgate* tipo jutikliai turi mažesnį didžiausią galimą diskretizavimo dažnį, bet jais nuskaitytame signale yra mažiau triukšmo. Nuskaityti signalai pateikti 22 paveiksle.



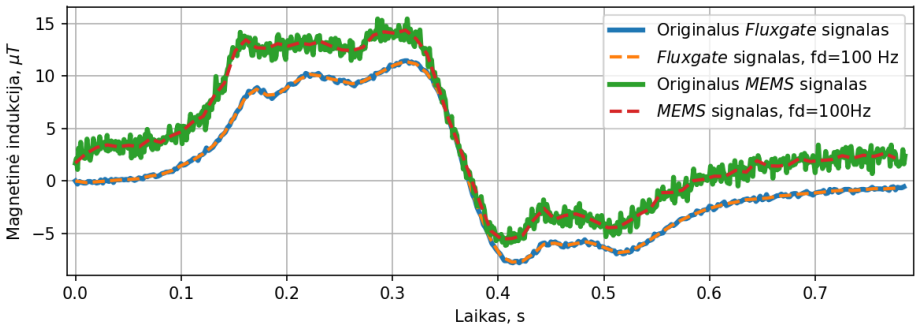
22 pav. Iš magnetinių jutiklių nuskaitytas magnetinės indukcijos kitimas laike

Tam, kad diskretizavimo dažnis būtų vienodas ir būtų galima signalus palyginti, MEMS tipo jutiklių signalas yra filtruojamas 250 Hz žemų dažnių filtru tam, kad po diskretizavimo dažnio sumažinimo nebūtų dažnių persidengimo ir tada jo imčių skaičius sumažinamas 2 kartus. Kaip galima matyti 22 ir 23 paveiksluose, abiem jutiklių tipais išmatuojami magnetiniai laukai kinta vienodai, vienintelis skirtumas – triukšmo lygis signale.



23 pav. Magnetinių jutiklių signalai ir jų spektrai

Pagal signalo spektrą (žr. 23 pav.) galima matyti, kad visa naudinga signalo informacija yra žemuose dažniuose iki 30 Hz, o toliau – triukšmas. Tai reiškia, kad signalo diskretizavimo dažnį galima sumažinti dar 5 kartus, t.y. 100 Hz diskretizavimo dažnis yra pakankamas nuskaityti tokiam signalui. 24 paveiksle pateikti signalų, su sumažintu diskretizavimo dažniu, pavyzdžiai.



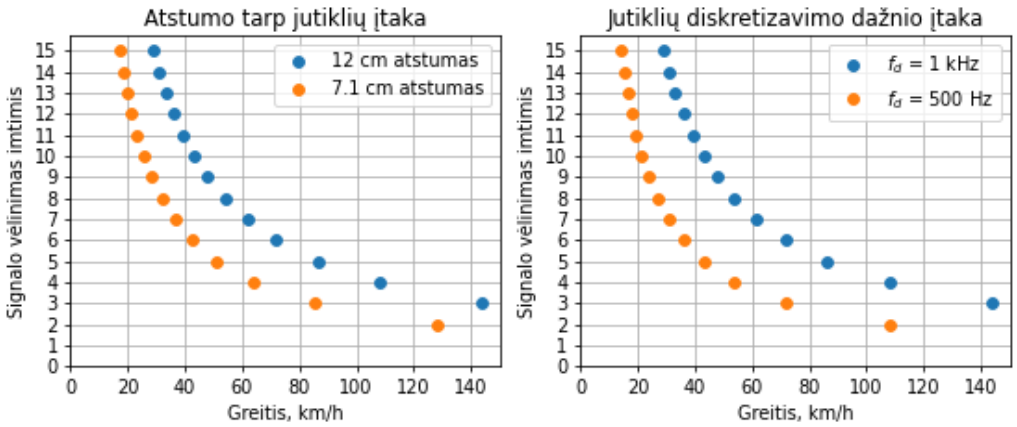
24 pav. Originalūs signalai ir signalai su diskretizavimo dažniu, sumažintu iki 100 Hz

Suvienodinus diskretizavimo dažnį iki 100 Hz, galima matyti, kad signalai yra tokie patys, bet su keliais skirtumais – MEMS magnetinio jutiklio signale yra daugiau triukšmo ir, lyginant su *fluxgate* jutikliu, signalas turi nuolatinę dedamąją bei yra šiek tiek pasislinkęs laike. Nuolatinė dedamoji ir poslinkis matomas dėl to, kad abiejų tipų jutikliai yra skirtingose paviršinio montažo plokštės vietose. Taigi duomenų analizei ir klasifikavimui bus naudojami *fluxgate* tipo jutiklio signalai dėl mažesnio skaičiavimų kiekio ir mažesnio triukšmo signale.

Panašiose sistemose, aprašytose 1.2 skyriuje, antras magnetinis jutiklis įprastai yra naudojamas tam, kad būtų galima nustatyti įvairių TP judėjimo greitį. Literatūros šaltiniuose skaičiavimams yra naudojami X, Y ir Z ašių signalai, kurie dažniausiai yra sujungiami į vieną bendrą signalą paskaičiuojant jų vektoriaus ilgį. Signalas, gautas šiam vektoriaus ilgiui kintant laike vadinamas magnetiniu parašu. Apskaičiavus koreliaciją tarp magnetinių parašų iš dviejų magnetinių jutiklių, TP greitis yra apskaičiuojamas pasinaudojant (1) formule:

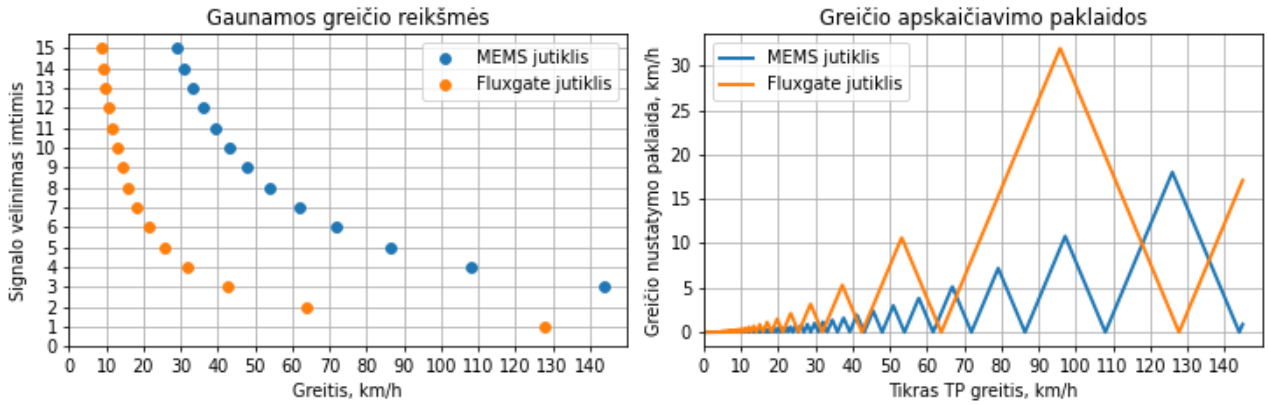
$$V = \frac{d \times f_d}{N}, \quad (1)$$

kur V yra greitis (m/s), d yra atstumas tarp magnetinių jutiklių (m), N – signalo vėlinimas, gautas apskaičiavus signalų koreliaciją (atskaitomis), o f_d yra diskretizavimo dažnis (Hz). Iš (1) formulės galima pastebėti, kad greičio skaičiavimo tikslumas priklausys nuo jutiklio diskretizavimo dažnio ir nuo atstumo tarp jutiklių. Dėl šios priežasties MEMS tipo jutiklis yra labiau tinkamas skaičiuoti greičiui negu *fluxgate* tipo jutiklis - jo diskretizavimo dažnis yra didesnis (MEMS – 1 kHz, o *fluxgate* – 500 Hz) ir atstumas tarp abiejų to pačio tipo jutiklių yra didesnis (MEMS – 120 mm, o *fluxgate* – 71 mm). 25 paveiksle pateikti grafikai, vaizduojantys kokią įtaką galimoms greičio reikšmėms turi atstumas tarp jutiklių ir jų diskretizavimo dažnis.



25 pav. Signalo vėlinimo imtimis priklausomybė nuo greičio esant skirtingiems atstumams tarp jutiklių ir skirtingiems diskretizavimo dažniams

Kaip galima matyti 25 paveiksle, didėjant TP judėjimo greičiui, gaunama koreliacijos reikšmė mažėja netiesiškai – galimos koreliacijos reikšmės yra sveiki skaičiai ir dėl šios priežasties galimi užfiksuoti greičiai yra labai riboti. Kuo didesnis greitis, tuo didesnis yra atstumas iki sekančio taško, tai reiškia, kad didėjant greičiui didėja ir galima matavimo paklaida. Norint ją sumažinti reikia, didinti diskretizavimo dažnį ir / arba atstumą tarp jutiklių. Pagal sistemos parametrus, aprašytus 2 skyriuje, apskaičiuotos galimos greičio reikšmės ir paklaidos kiekvieno tipo jutikliui pateiktos 26 paveiksle.

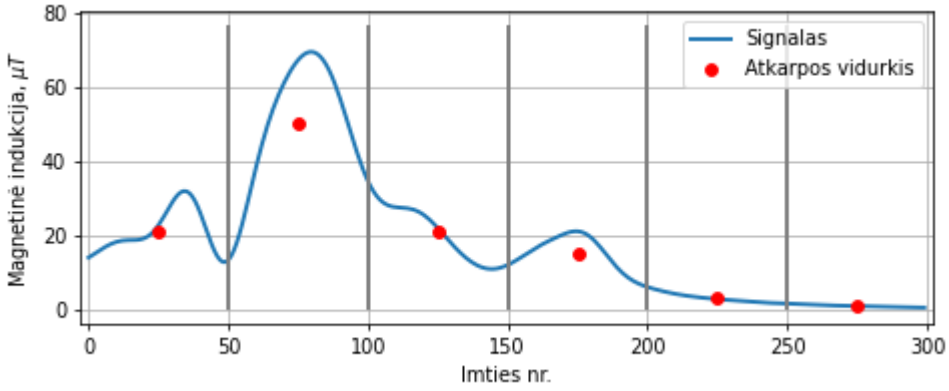


26 pav. Sistemoje esančių magnetinių jutiklių galimos užfiksuoti greičio reikšmės ir skaičiavimo paklaidos

Galima matyti, kad MEMS jutikliu greitis apskaičiuojamas daugmaž du kartus tiksliau – pavyzdžiui, po koreliacijos skaičiavimo *fluxgate* jutikliu, visi greičiai nuo apytiksliai 63 km/h iki 97 km/h bus apskaičiuoti kaip 63 km/h. Jeigu TP važiuos 97 km/h greičiu, skaičiavimo paklaida bus net 32 km/h. Tuo tarpu naudojant MEMS jutiklį, didžiausia paklaida šiame greičio ruože siektų 11 km/h dėl to, kad sistemoje tarp šio tipo magnetinių jutiklių yra didesnis atstumas ir jų diskretizavimo dažnis yra didesnis.

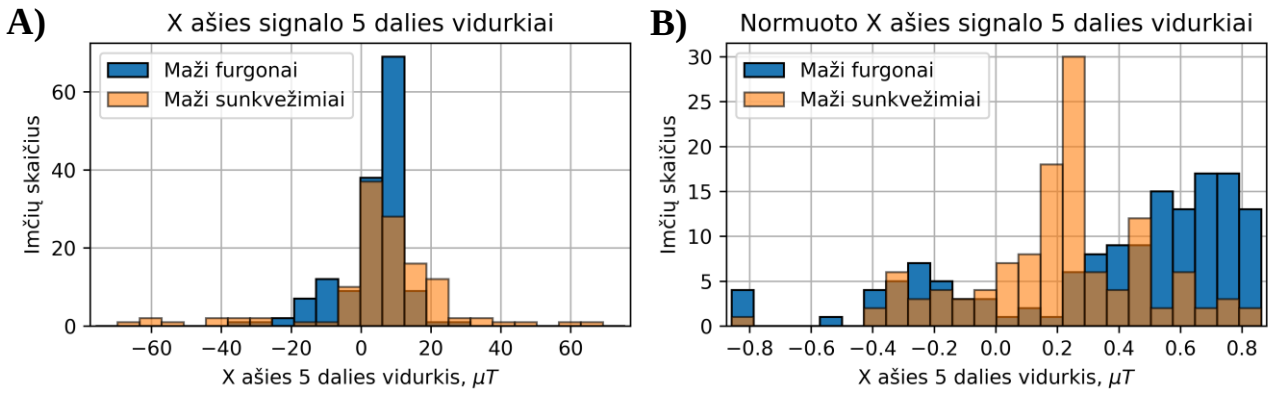
Greičio nustatymo paklaidą galima sumažinti padidinant magnetinių jutiklių duomenų kiekį. Tai padaryti galima interpoliuojant turimus duomenis, taip dirbtinai padidinant diskretizavimo dažnį, arba tiesiogiai padidinti diskretizavimo dažnį duomenų surinkimo sistemoje. Abiem atvejais padidėtų skaičiavimų kiekis, bet tuo pačiu pagerėtų ir rezultatai dėl mažesnių paklaidų (žr. 25 ir 26 pav.). Dar vienas būdas padidinti greičio skaičiavimo tikslumą yra padidinti atstumą tarp dviejų jutiklių – šiuo būdu keičiasi sistemos dydis, bet skaičiavimų kiekis lieka toks pats. Vis dėlto, atstumui tarp jutiklių yra ribos – jeigu tarpas tarp jutiklių yra pakankamai didelis, TP lėtėjant arba greitėjant signalas abiejuose jutikliuose būtų skirtingos formos dėl skirtingo TP greičio signalo nuskaitymo metu, o tai apsunkintų signalų apdorojimą ir koreliacijos skaičiavimą.

Greitis yra svarbus parametras nustatant TP ilgį, pagal kurį galima tiesiogiai atlikti klasifikavimą. Vis dėlto, yra kitų klasifikavimo metodų, kuriems TP judėjimo greitis yra nesvarbus. Pavyzdžiui, [6] šaltinyje aprašomas klasifikavimo metodas, kuriame magnetinis signalas yra sudalinamas į kelias dalis ir iš kiekvienos iš jų apskaičiuotas parametras naudojamas kaip atskiras požymis. Straipsnyje šis parametras buvo kiekvienos atkarpos magnetinės indukcijos vidurkis. Sudalinto TP magnetinio parašo, kartu su atitinkamų atkarpų vidurkiais, pavyzdys pateiktas 27 paveiksle. Šį metodą taikyti galima magnetiniam signalui iš X, Y ir Z ašių bei jų vektoriaus ilgio, taip gaunant dar daugiau požymių klasifikavimui. Visi šie požymiai yra skirti įvertinti TP iškraipomą magnetinį lauką, kuris priklauso nuo TP kėbulo formos ir kitų dalių.



27 pav. Magnetinis parašas, padalintas į 6 dalis ir kiekvienos jų magnetinės indukcijos vidurkis

TP pravažiuojant virš jutiklių bloko gaunamas magnetinis parašas skiriasi amplitude, priklausomai nuo atstumo, kuriuo TP pravažiuoja nuo jutiklių bloko. Dėl šios priežasties prieš signalo sudalinimą ir atkarpų vidurkių skaičiavimą reikalingas papildomas žingsnis – normavimas. 28 paveiksle pavaizduoti rezultatai gauti vidurkį sunormavus ir palikus nenormuotą.



28 pav. Normuotos (A) ir nenormuotos (B) penktos dalies vidurkiai mažiems furgonams ir sunkvežimiams

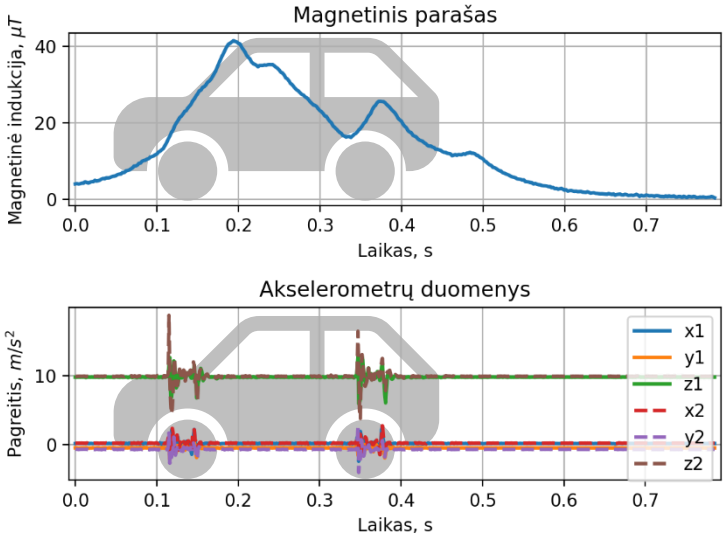
Kaip galima matyti paveiksle, prieš vidurkio normavimą buvo tik keli siauri intervalai, kuriuose didžioji dauguma imčių priklausė vienai klasei, o po normavimo išsiskyrė keli tokie intervalai. Šiuo atveju (žr. 28 pav. B dalį) intervale nuo 0,6 iki 1 yra 85 % tikimybė, kad TP yra mažas furgonas, o intervale nuo 0 iki 0,3 yra 86 % tikimybė, kad TP yra mažas sunkvežimis. Visi požymiai gauti pagal [6] šaltinio metodiką buvo panašūs į 28 paveikslo požymius arba prastesni, taigi norint išskirti TP su didesne tikimybe reikia kartu panaudoti kitais metodais gautus požymius.

Literatūros šaltiniuose buvo ir kitų dažnai naudojamų požymių, tokių kaip įvairūs pirmo ir antro laipsnio išvestinių parametrai, signalų energija, vidurkis, didžiausios ir mažiausios reikšmės bei nulių kirtimo vietos. Apskaičiavus tokius požymius ir išanalizavus skirtumus tarp skirtingų klasių buvo pastebėta, kad dauguma jų yra gerokai prastesni negu požymiai, gauti sudalinus signalą ir apskaičiavus gautų dalių vidurkius. Taigi, [6] ir [9] šaltinių bei atliktos analizės rezultatai parodo, kad skirtingų klasių TP sukelia skirtingus magnetinio lauko iškraipymus. Dar vienas parametras, naudojamas daugumoje literatūros šaltinių yra TP ilgis. Šiam parametrui apskaičiuoti reikalingas TP greitis, kurį galima apskaičiuoti pagal (1) formulę. TP ilgis yra požymis, kurį galima panaudoti atskiriant mažas TP nuo didesnių, pavyzdžiui, mažą dvivietę TP nuo sunkvežimio [15]. Pilnas apskaičiuotų požymių, kurių dalis buvo išgauti pasiremiant literatūros šaltiniais, sąrašas pateiktas 1 priede.

Išanalizavus magnetinių jutiklių duomenis buvo pastebėta, kad pasinaudojant tik iš jų duomenų apskaičiuotais parametrais nėra įmanoma su 100 % tikimybe atskirti net 2 klasių TP. Literatūros šaltiniuose klasifikavimui pagal magnetinių jutiklių duomenis dažniausiai naudojamas dirbtinis intelektas – juo pasinaudojant galima pagreitinti sąsajų tarp požymių atradimą, o šios sąsajos yra reikalingos norint patikimai atlikti TP klasifikavimą. Magnetiniai jutikliai gali būti panaudoti ne tik požymių apskaičiavimui – juos galima panaudoti greičio matavimui, kuris gali būti panaudotas kitiems skaičiavimams.

3.2. Akselerometrai

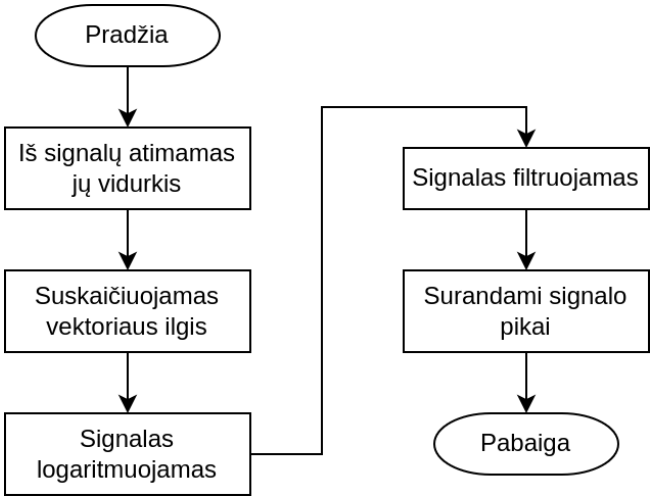
Duomenų surinkimo sistemoje buvo naudojami du akselerometrai, kurių diskretizavimo dažnis yra 1 kHz. Vieno pravažiavimo metu nuskaityti duomenys iš magnetinio jutiklio ir akselerometrų pavaizduoti 29 paveiksle.



29 pav. Nuskaityti akselerometrų duomenys kartu su tos pačios TP magnetiniu parašu

Akselerometrų signale matomi du laiko intervalai, kai vibracijos yra didesnės negu kitais laiko intervalais – šių intervalų skaičius sutampa su TP ašių skaičiumi ir jie atsiranda tuo metu, kai TP važiuoja virš dviejų 2 skyriuje aprašytų griovelių. TP ašių skaičius yra tinkamas požymis klasifikavimui pagal TP ašių skaičių ir buvo naudotas kaip vienas iš kelių požymių ar net kaip vienintelis požymis keliuose metoduose, aprašytuose 1 skyriuje. Kaip galima matyti 29 paveiksle, pirmo ir antro akselerometrų signalai sutampa, taigi tolimesniuose tyrimuose bus naudojamas tik vienas akselerometras, nes antro akselerometro duomenys jokios papildomos informacijos neprideda.

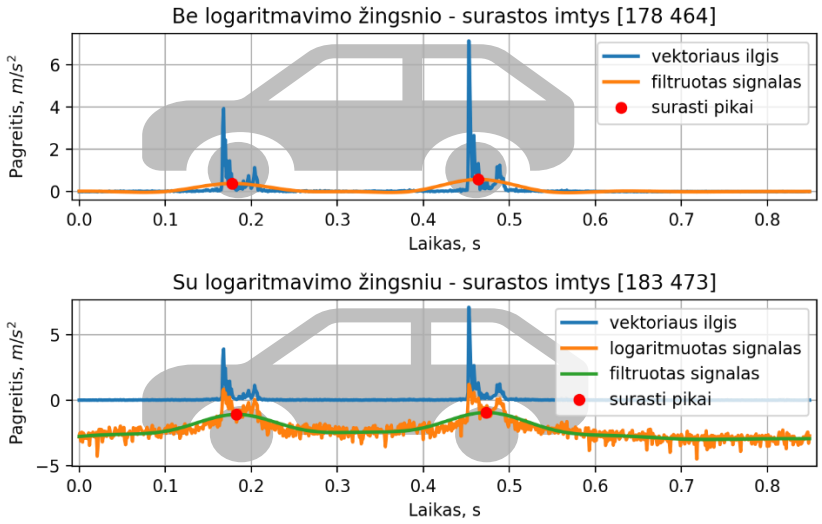
Tam, kad būtų galima ašių skaičių panaudoti klasifikavimui ir kitiems skaičiavimams, reikia signale surasti tas vietas, kur TP ašys turi įtakos signalui. Šios užduoties sprendimo algoritmas pateiktas 30 paveiksle.



30 pav. TP ašių skaičiaus ir vietos signale suradimo algoritmas

Pirmiausia iš kiekvieno signalo yra atimamas vidurkis – tokiu būdu yra kompensuojama gravitacijos dedamoji, kuri labiausiai veikia Z ašį, bet jei jutiklis padėtas nevisai lygiai, tai ji yra juntama ir kitose ašyse. Toliau skaičiuojamas vektoriaus ilgis iš visų akselerometro ašių tam, kad skaičiavimuose būtų

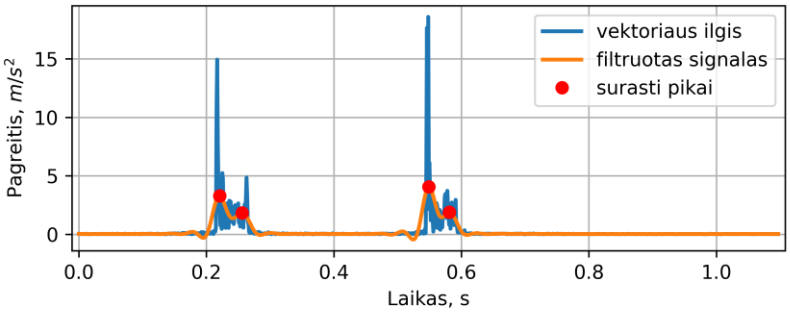
atsižvelgta į vibracijas visose trijose ašyse. Sekantis žingsnis yra logaritmavimas. Šis žingsnis nėra privalomas, bet padeda tiksliau nustatyti ašių vietą laike – signalo logaritmavimas padeda nuslopinti signalo vietas su staigiais didelių amplitudžių pokyčiais, bet tuo pačiu sustiprina vietas, kur signalas kinta lėčiau. Toliau signalas yra filtruojamas žemų dažnių filtru tam, kad jame liktų tik žemiausio dažnio dedamosios. Filtro dažnis buvo eksperimentiškai parinktas 7 Hz, nes tada geriausiai nuslopinamos aukštesniųjų dažnių dedamosios ir geriausiai išryškėja pikas ties sustiprėjusiomis vibracijomis. Tai yra svarbu tam, kad būtų galima atskirti ašis, kurios yra labai arti viena kitos, pavyzdžiui, sunkvežimių su trimis ašimis atveju. Galiausiai apdorojimų rezultate surandami pikai, kurie atitinka TP ašių vietas laike. Algoritmo panaudojimo rezultatas pateiktas 31 paveiksle. [16]



31 pav. TP ašių ieškojimo signale rezultatai

Kaip galima matyti grafikuose, logaritmavimo žingsnis stipriai nuslopino didelės amplitudės vibracijas, esančias vibracijų intervalo pradžioje. Dėl šios priežasties ašių vieta signale buvo nustatyta tiksliau negu nenaudojant logaritmavimo žingsnio (surastas pikas buvo arčiau didelės amplitudės vibracijų intervalo centro), o tai yra gana svarbu dėl to, kad pasitaiko tokių signalų, kad šio vibracijų sustiprėjimo visai nebūna arba jis būna skirtingas kiekvienai ašiai.

Analogiškas algoritmas gali būti panaudotas TP greičio suskaičiavimui pagal akselerometrų signalus. Šiuo atveju reikia panaudoti aukštesnio dažnio ŽDF. Eksperimentiškai buvo nustatyta, kad tinkamiausias dažnis šiai užduočiai yra 30 Hz, nes tada pakankamai ryškiai pasimato kiekvienas reikalingas pikas. Kiekviena ašis turi du pikus dėl to, kad ant kelio yra du grioveliai – kiekvieną kartą TP pravažiavus virš griovelio su viena iš ašių gaunamas vienas pikas. Signale aptikus visus reikalingus pikus greitis apskaičiuojamas pagal (1) formulę vietoj atstumo tarp magnetinių jutiklių naudojant atstumą tarp griovelių, o vietoj signalo vėlinimo – laiko skirtumą tarp gretimų pikų, priklausančių tai pačiai ašiai. Algoritmo surasti pikai dviejų ašių TP pavaizduoti 32 paveiksle. [16]



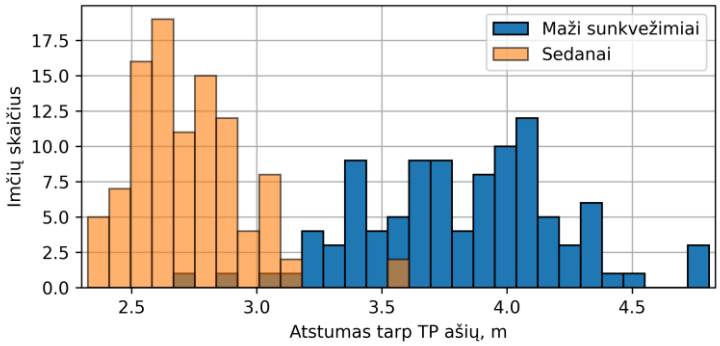
32 pav. Greičio skaičiavimui surasti pikai signaluose

Kiekvienai ašiai yra surandamas atskiras greitis, taigi skaičiavimų pabaigoje yra apskaičiuojamas suskaičiuotų greičių vidurkis ir šis vidurkis yra laikomas TP judėjimo greičiu.

Žinant TP greitį, pagal surastas TP ašių vietas signale galima apskaičiuoti atstumą tarp TP ašių:

$$l = V \times t, \tag{2}$$

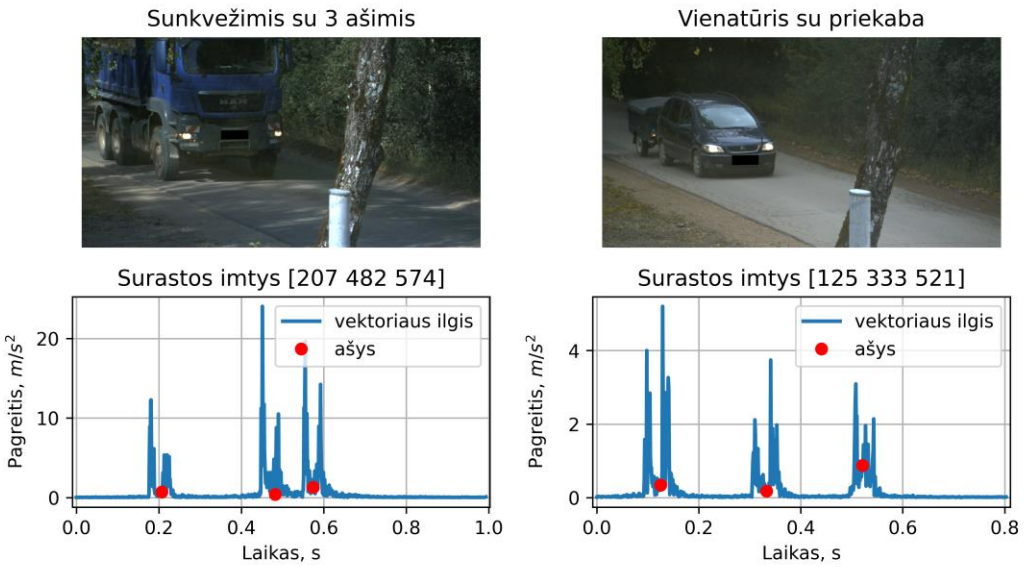
kur l yra atstumas tarp ašių, V yra greitis, o t – laikas tarp signale aptiktų ašių vietų, apskaičiuojamas. Atstumas tarp ašių gali būti panaudotas kaip požymis klasifikavimui, pavyzdžiui, 33 paveiksle pavaizduoti šie atstumai apskaičiuoti paėmus po 100 pravažiavimų iš dviejų skirtingų klasių:



33 pav. Sedanų ir mažų sunkvežimių atstumo tarp ašių palyginimas

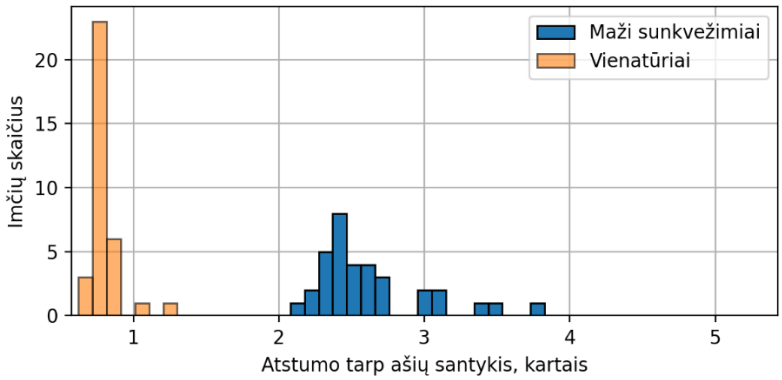
Galima pastebėti, kad atstumas tarp sedanų ašių yra vidutiniškai mažesnis negu atstumas tarp mažų sunkvežimių ašių. Vis dėlto, galimos ribos persidengia – nesvarbu kur būtų nustatytas slenkstis, visada pasitaikytų atvejų, kai apskaičiuotas ilgis būtų priskirtas ne tai TP klasei. Pavyzdžiui, parinkus, kad TP, kurių atstumas tarp ašių yra mažesnis negu 3,2 m būtų priskirtos sedano klasei, o visos kitos – mažų sunkvežimių klasei, 98 % sedanų būtų suklasifikuoti teisingai, o mažų sunkvežimių - 97 %.

Virš jutiklių bloko pravažiavus TP su priekaba yra užfiksuojamos trys ašys ir TP įprastai būtų priskirta vienintelei klasei kuri turi tris ašis – sunkvežimiams su 3 ašimis. Vis dėlto, pasinaudojant atstumais tarp aptiktų ašių galima atskirti sunkvežimį su 3 ašimis nuo lengvos TP su priekaba, tokio atvejo pavyzdys pateiktas 34 paveiksle.



34 pav. Dviejų skirtingų tipų TP su trimis ašimis

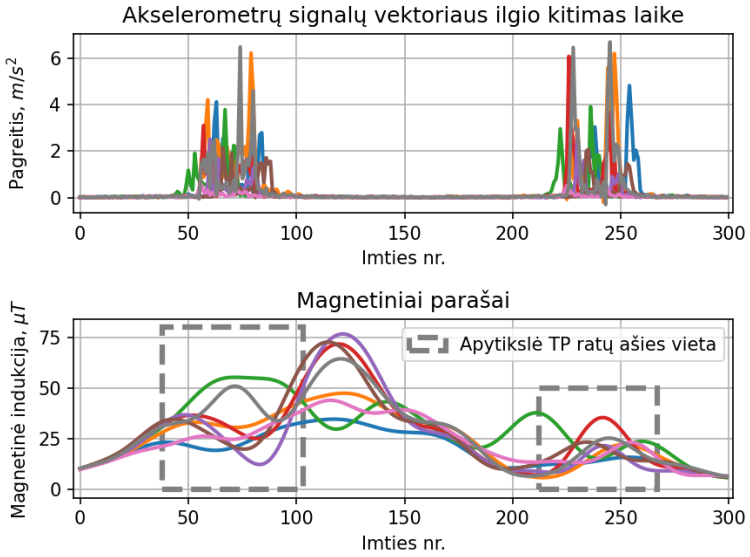
Kaip galima matyti 34 paveiksle, vienatūrės TP ašys nutolusios panašiu atstumu, o sunkvežimio dvi yra šalia viena kitos, o trečia nutolusi didesniu atstumu. Pasinaudojant atstumų pirma-antra ašis ir antra-trečia ašis santykiu, galima atskirti šias dvi TP klases. 35 paveiksle pavaizduoti rezultatai suskaičiavus šiuos santykius 34 atvejams iš abiejų klasių.



35 pav. Atstumų tarp ašių santykiai sunkvežimiams su 3 ašimis ir vienatūriams su priekabomis

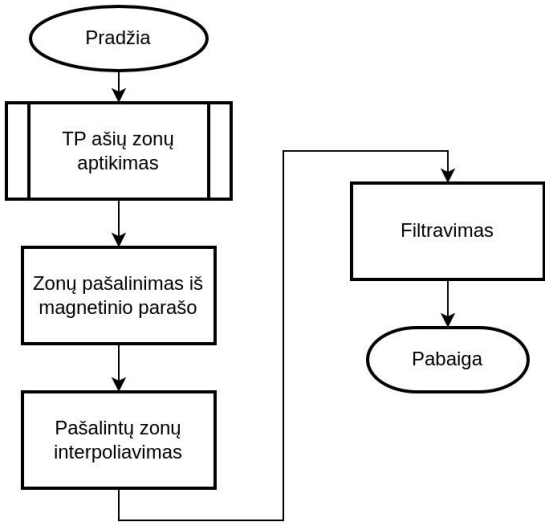
Kaip galima matyti histogramoje (žr. 35 pav.), pasinaudojant atstumų tarp ašių santykiu visada galima atskirti sunkvežimį nuo lengvos TP, nes sunkvežimiams santykis yra virš 2 kartų, o lengvoms TP – mažiau nei 1,5 karto.

TP važiuojant virš jutiklių bloko su ratais, jei jie turi magnetinių savybių, magnetinis parašas yra iškraipomas kiekvieną kartą vis skirtingai, priklausomai nuo rato pozicijos jutiklių bloko atžvilgiu. Tokie nenuspėjami iškraipymai gali turėti neigiamą įtaką duomenų apdorojimui ir klasifikavimui. Magnetiniai parašai, gauti pravažiavus tai pačiai TP kelis kartus pavaizduoti 36 paveiksle.



36 pav. Ratų įtaka magnetiniam parašui kelis kartus pravažiavus tai pačiai TP

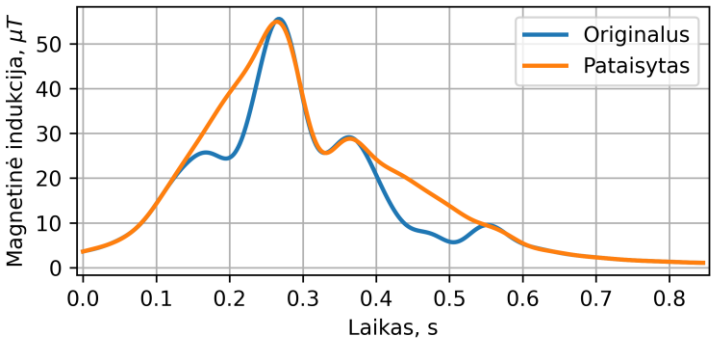
Kaip galima matyti, ratų įtaka magnetiniam parašui kiekvieną kartą yra vis kitokia. Šis iškraipymas gali viršyti net pačios TP magnetinio parašo lygį, tokiu atveju atliekant normavimą požymių skaičiavimams, kurie aprašyti 3.1 skyriuje, būtų gauti prastesni rezultatai. Sudalinus tokį magnetinį parašą į kelias dalis, tose dalyse, kuriose yra šie ratų iškraipymai, apskaičiuoti vidurkiai būtų netikslūs dėl nenuspėjamų iškraipymų. Šios problemos išvengti galima panaikinant iškraipytas vietas iš magnetinio parašo. Kaip galima pastebėti 36 paveiksle, magnetinio parašo iškraipymų vietos sutampa su vietomis akselerometro signale, kuriose galima aptikti TP ašis. Pasinaudojant šia informacija magnetinius parašus galima pataisyti – pilnas algoritmas pateiktas 37 paveiksle.



37 pav. Magnetinio parašo korekcijos algoritmas

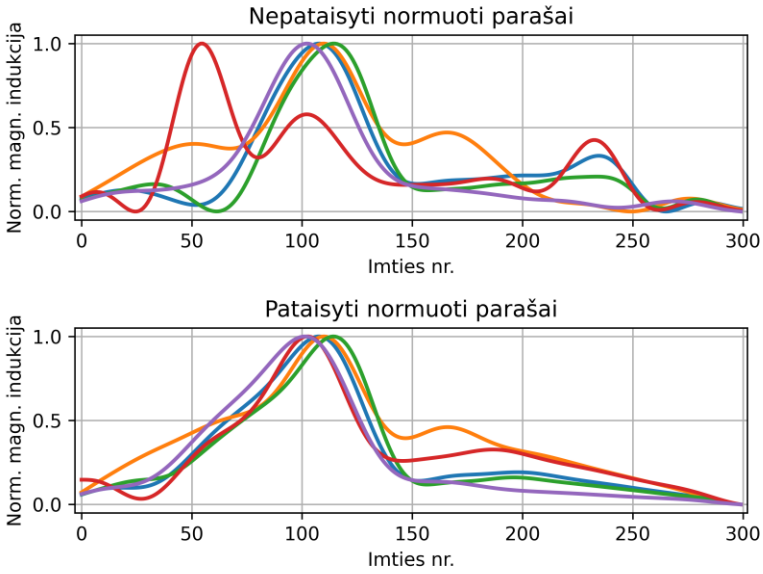
Pirmiausia panaudojama ašių aptikimo algoritmu (žr. 30 pav.), kuriuo gaunamos zonos, kuriose ratai gali turėti įtaką magnetiniam parašui. Toliau šios zonos yra panaikinamos iš parašo ir tiesiškai interpoliuojamos. Tam, kad būtų panaikinti staigūs pokyčiai signale, atsiradę dėl tiesinio interpoliavimo, signalas yra filtruojamas 30 Hz ŽDF. Toks dažnis buvo parinktas dėl to, kad

magnetiniame paraše virš šio dažnio nėra naudingos informacijos (žr. 3.1 skyrių). Rezultatas, gautas pritaikius algoritmą, pavaizduotas 38 paveiksle.



38 pav. Magnetinio parašo korekcijos algoritmo rezultatas

Kaip galima matyti paveiksle, korekcijos metu buvo pašalinta TP ratų įtaka magnetiniam parašui. Tokį parašą jau galima sėkmingai panaudoti parametrų skaičiavimui ir analizei, nes jame nebėra atsitiktinių iškraipymų. Analizuojant magnetinius parašus buvo pastebėta, kad jei TP ratai magnetinių savybių neturi arba buvo per toli, kad iškraipytų magnetinį lauką, šis algoritmas magnetinio parašo beveik nepakeičia – tose vietose kur ratai turėtų įtaką magnetiniam parašui, jis kinta tolygiai, taip pat kaip gaunama pritaikius algoritmą. Pataisius visus tos pačios TP magnetinius parašus jie pasidaro gerokai panašesni (žr. 39 pav.).



39 pav. Tos pačios TP pataisyti ir originalūs magnetiniai parašai, gauti jai pravažiavus kelis kartus

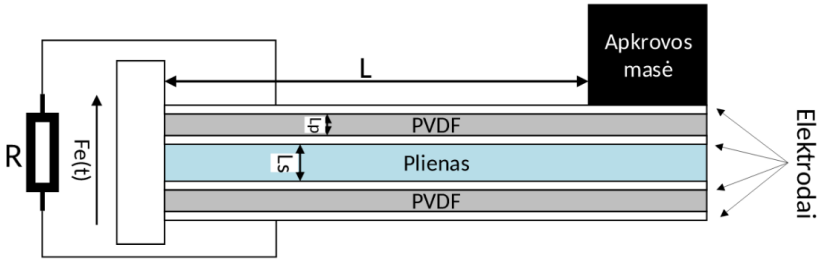
39 paveiksle pateikti tie patys magnetiniai parašai kaip 36 paveiksle tik normuoti. Kaip galima matyti, nepritaikius korekcijos algoritmo ir atlikus normavimą magnetiniai parašai yra gana skirtingi, ypač išsiskiria tos vietos, kuriose yra iškraipymų dėl TP ratų. Pataisius magnetinius parašus jie pasidaro panašesni negu prieš pataisymą. Vis dėlto, net ir po korekcijos algoritmo pritaikymo, tos pačios TP skirtingų pravažiavimų magnetiniai parašai šiek tiek skiriasi. Taip yra dėl to, kad kiekvieną kartą TP

važiuojant virš jutiklių bloko būna išmatuojama vis kita magnetinio lauko vieta, o jis nėra tolygus per visą TP plotą.

Taigi, duomenis, surinktus pasinaudojant akselerometrais, galima panaudoti įvairiais būdais – apskaičiuoti TP greitį bei ilgį, suskaičiuoti TP ašių kiekį ir atstumus tarp jų ir šiuos duomenis panaudoti klasifikavimui. Taip pat pasinaudojant akselerometrų duomenimis galima panaikinti magnetinių jutiklių duomenyse esančius nepastovius iškraipymus atsirandančius dėl TP ratų įtakos.

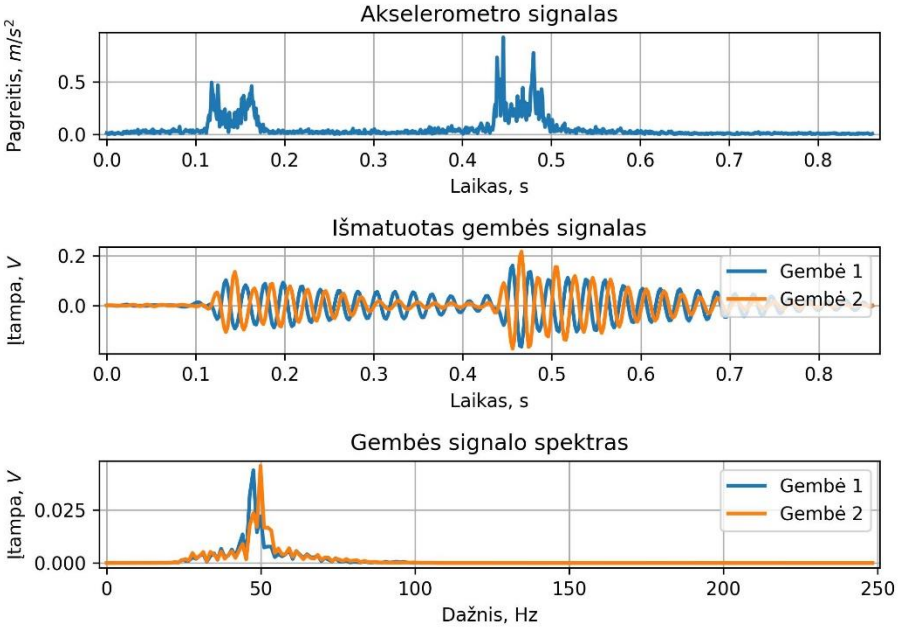
3.3. Pjezoelektrinės gembės

Jutiklių bloke buvo naudojamos dvi pjezoelektrinės gembės. Gembė yra sudaryta iš plieninės šerdies, kurios abiejose pusėse yra pritvirtinta PVDF plėvelė. Šios konstrukcijos vienas galas yra įtvirtintas energijos surinkimo plokštėje, o prie kito galo pritvirtinta apkrovos masė. Gembės struktūra pateikta 40 paveiksle.



40 pav. Gembės struktūra

Gembei lankstantis pjezoelektrikas sukuria įtampą, kurią matuojant galima įvertinti gembės išlinkimą. TP važiuojant virš sistemos būna sukeliamos vibracijos, kurios virpina gembę, taigi matuojant pjezoelektrinės gembės įtampą galima įvertinti TP sukeltas vibracijas. Signalai, gauti pravažiavus vienai TP pavaizduoti 41 paveiksle.



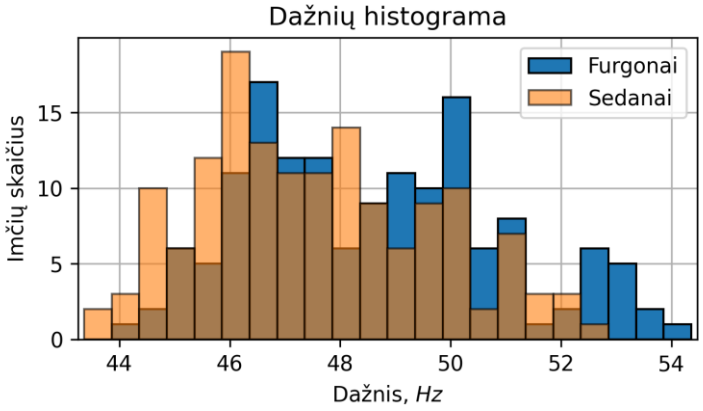
41 pav. Išmatuotas gembės signalas ir jo spektras

Kaip galima pastebėti, gembių vibracijų pradžia sutampa su akselerometro vibracijų pradžia. Taip yra dėl to, kad didžiausios vibracijos būna sukeliamos tuo metu, kai TP pravažiuoja per vieną iš griovelių, taigi, pjezoelektrinių gembių signalas būtų tinkamas kaip alternatyva akselerometrams TP ašių kiekio skaičiavimui. Gembių signalo spektre galima matyti, kad gembės vibruoja daugmaž 50 Hz dažniu – toks dažnis yra jų rezonansinis dažnis, kuris priklauso nuo masės ant gembės galo bei pačios gembės parametrų, tokių kaip jos ilgis, plotis ir šerdies medžiaga. Antrosios gembės rezonansinis dažnis yra aukštesnis, todėl ji visada po sužadinimo vibruoja didesniu dažniu.

Skirtumų ir požymių ieškojimui gembių signaluose pirmiausia buvo atskirtos 2 TP klasės, turinčios 2 ašis – furgonai ir sedanai (pavyzdžiai pateikti 42 paveiksle).

42 pav. Lyginamos TP klasės

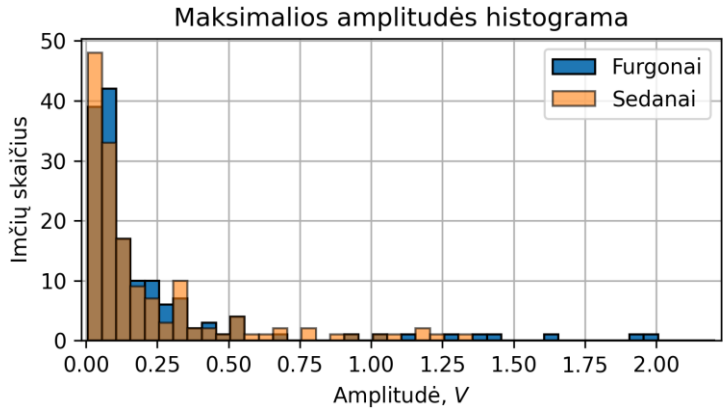
Pradiniams skirtumų ir požymių ieškojimams šios klasės buvo atrinktos dėl to, kad jos yra skirtingos kėbulo matmenimis ir mase. Iš kiekvienos klasės atrinkus po 150 pravažiavimų buvo palyginti keli parametrai, gauti iš gembės sugeneruoto signalo spektro – didžiausios amplitudės spektrinės dedamosios dažnis (žr. 43 pav.) ir amplitudė (žr. 44 pav.) bei šių parametrų priklausomybės nuo TP greičio (žr. 45 pav.).



43 pav. Gembių didžiausios amplitudės vibracijų dažniai po TP pravažiavimo

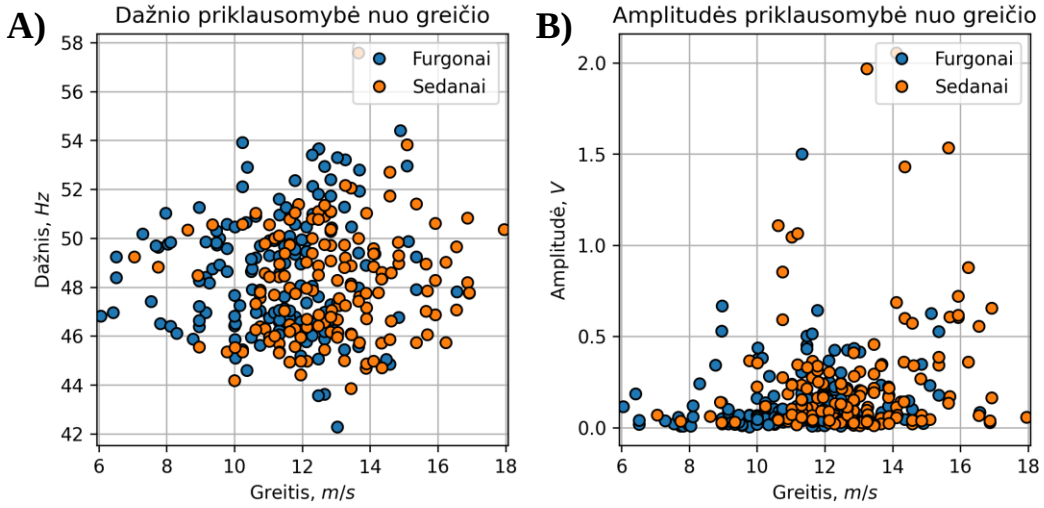
Kaip galima matyti didžiausios amplitudės vibracijų dažnių histogramoje (žr. 43 pav), vibravimo dažnis negalėtų būti tinkamas požymis klasifikavimui, nes pravažiavus TP iš bet kurios iš šių dviejų klasių, gembės vibruoja panašiu dažniu kuris abiem atvejais yra intervale nuo 42 Hz iki 54 Hz.

Kadangi abiejų klasių aibės sutampa ir nėra intervalų, kuriuose jos išsiskirtų – šis parametras negali būti laikomas požymiu klasifikavimui.



44 pav. Gembių vibravimo signalo didžiausios spektrinės dedamosios amplitudės po TP pravažiavimo

Taip pat, kaip ir dažnio atveju, 44 paveiksle galima matyti, kad gembių vibracijų amplitudė tinkamas požymis dėl to, kad abiejų klasių aibės sutampa.



45 pav. Gembės vibracijų dažnio (A) ir amplitudės (B) priklausomybės nuo greičio

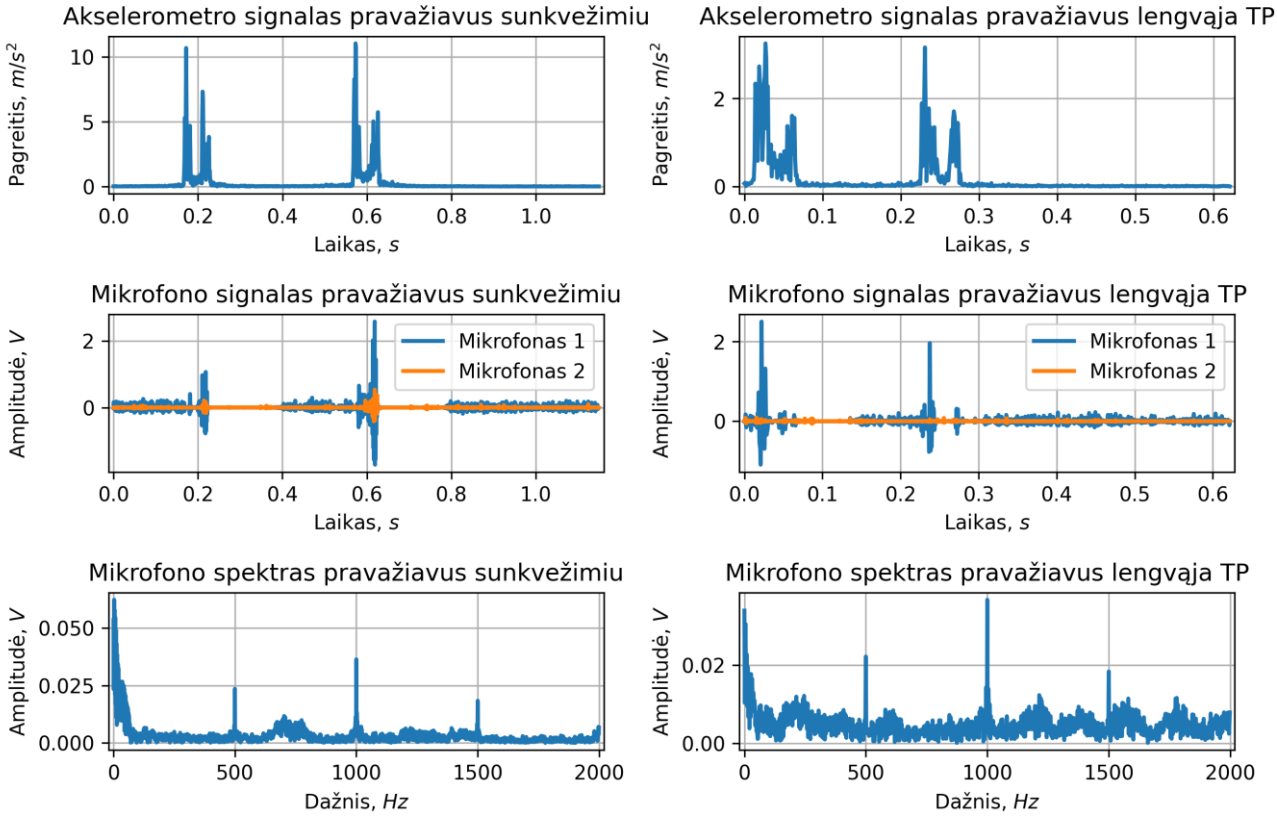
Didžiausios gembių vibracijų amplitudės yra sužadinamos TP važiuojant per griovelį kelyje, o važiuojant didesniu greičiu turėtų būti sužadinamos didesnės vibracijos dėl stipresnio smūgio. Dėl to buvo nuspręsta patikrinti ar gembės signalo didžiausios spektro dedamosios dažnis ir amplitudė priklauso nuo TP greičio. Vis dėlto, kaip galima matyti 45 paveiksle, tarp šių parametrų jokios priklausomybės nėra. Kitas parametras nuo kurio galėtų priklausyti vibracijų amplitudė ir dažnis yra TP masė, bet, bandymų metu tiksli pravažiavusių TP masė nebuvo žinoma.

Atlikus bandymus su kitomis TP klasių kombinacijomis ir gavus panašius rezultatus buvo nuspręsta, kad pjezoelektrinės gembės nėra tinkamas jutiklis TP klasifikavimui. Esant poreikiui, pjezoelektrines gembes būtų galima panaudoti vietoj akselerometrų TP ašių skaičiaus ir vietos signale nustatymui ir šią informaciją panaudoti klasifikavimui. Lyginant su akselerometrais, gembių signalas yra mažiau tinkamas TP ašių vietos nustatymui dėl to, kad akselerometro signale vibracijos po pravažiavimo per

griovelį būna sužadinamos ir nuslopsta daug greičiau nei gembių signale. Taip yra dėl to, kad gembės turi didesnę masę nei matavimo elementas MEMS akselerometruose ir dėl didesnės inercijos jos reaguoja gerokai lėčiau į išorines jėgas, bet po to ilgiau virpa savo rezonansiniu dažniu.

3.4. Mikrofonai

Kiekviena TP važiuodama skleidžia skirtingus garsus dėl padangų protektoriaus, variklio veikimo ir dėl kitų įvairių priežasčių. Tam, kad šiuos garsus būtų galima įrašyti, jutiklių bloke buvo panaudoti mikrofonai. Mikrofonai buvo du dėl to, kad pasinaudojant dviem mikrofonais galima nustatyti TP judėjimo kryptį ir taip pat atskirti kelių tipų TP [11]. Vis dėlto, šio projekto duomenų surinkimo sistema skiriasi tuo, kad ji yra integruota po keliu, o [11] šaltinio sistema buvo įdiegta virš kelio. 46 paveiksle pavaizduoti akselerometrų ir mikrofonų duomenys, nuskaityti pravažiavus sunkvežimiu ir lengvąja TP.



46 pav. Mikrofonų ir akselerometrų signalų palyginimas, kai virš sistemos pravažiavo sunkvežimis ir hečbekas

Kaip galima matyti 46 paveiksle, mikrofonų signale galima atskirti TP ašių skaičių ir vietas laike, taip pat kaip akselerometrų ir pjezoelektrinių gembių signaluose. Vis dėlto, mikrofono signalas šiems parametrams yra gerokai prastesnis šaltinis dėl to, kad pravažiuojant per abu griovelius ne visada gaunami 2 vibracijų sustiprėjimai. Mikrofonų signale taip pat galima pastebėti, kad tuo metu, kai TP yra tiesiai virš jutiklių bloko (signalo atkarpa tarp ašių), triukšmo lygis yra toks pats kaip ir tada, kai TP yra jau nutolusi (signalo galas – sunkvežimio signale nuo 0,8 s, o lengvosios TP signale nuo 0,4 s).

46 paveikslo spektruose didžiausias matomas skirtumas yra didesnė dažninių dedamųjų ties 750 Hz amplitudė, bet ši signalo savybė nėra unikali sunkvežimiams ir pasitaiko per retai, kad būtų tinkama kaip požymis klasifikavimui. Jokių kitų reikšmingų skirtumų nebuvo pastebėta ir palyginus kitas TP poras.

Paklausius garso įrašų, girdimas garsas nebuvo panašus į šalia kelio stovint girdimą TP skleidžiamą garsą. Taip pat nebuvo ir garso sustiprėjimo tuo metu, kai TP važiavo tiesiai virš jutiklių bloko – girdimas yra tik duslus garsas, sukeltas TP važiuojant per griovelius. Taip gali būti dėl to, kad, kaip rašoma [11] šaltinyje, TP skleidžiamas garsas patenka į ribas iki 10 kHz, o sistemos didžiausias įrašomas dažnis yra 2 kHz. Vis dėlto, šiame dažnių ruože vis tiek turėtų būti girdimas variklio apsukų ar padangų protektoriaus keliamas garsas važiuojant keliu, bet šie garsai po kelio danga yra per daug nuslopinti. Siekiant išgirsti šiuos garsus, buvo padidintas stiprinimas, bet tai situacijos nepagerino net pritaikius įvairius filtrus. Taip pat, abiejų mikrofonų signalai yra per daug skirtingi, kad juos būtų galima panaudoti TP važiavimo krypties nustatymui.

Atlikus mikrofonų signalų analizę buvo priimta, kad mikrofonų, įmontuotų į kelio dangą, signalai nėra tinkami TP klasifikavimui atlikti. Vienintelis požymis, kurį būtų galima apskaičiuoti iš mikrofonų signalo yra TP ašių skaičius ir atstumas tarp jų, bet šiai paskirčiai naudojant akselerometrų signalus gaunami rezultatai vis tiek būtų geresni.

3.5. Surinktų duomenų analizės apibendrinimas

Tyrimo metu buvo analizuojami duomenys iš 5 tipų jutiklių - 2 tipų magnetinių jutiklių, akselerometrų, pjezoelektrinių gembių ir mikrofonų. Pjezoelektrinės gembės ir mikrofonai neduoda papildomos informacijos klasifikavimui lyginant su akselerometrais, taigi jeigu sistemoje jau yra akselerometrai, nėra prasmės kartu naudoti ir pjezoelektrines gembes ar mikrofonus.

MEMS ir *fluxgate* magnetiniai jutikliai duoda panašius rezultatus, pagrindinis skirtumas – triukšmo lygis signale. Šią problemą galima spręsti signalą filtruojant, bet tada didėja skaičiavimų apimtis. Naudojant *fluxgate* tipo magnetinį jutiklį išmatuojamas signalas turi mažiau triukšmo, todėl šio jutiklio signalo galima nefiltruoti. Naudojant 2 magnetinius jutiklius greičio skaičiavimui yra svarbus diskretizavimo dažnis, o *fluxgate* tipo jutikliams, kurie buvo naudojami duomenų surinkimo sistemoje, didžiausias galimas yra 500 Hz. Taigi, greičio skaičiavimui MEMS jutikliai yra labiau tinkami, nes jais galima nuskaityti imtis didesniu dažniu taip gaunant didesnį greičio nustatymo tikslumą, o klasifikavimo parametrų skaičiavimui geresni *fluxgate* tipo magnetiniai jutikliai, nes jų signalas reikalauja mažiau papildomo apdorojimo ir jame yra mažesnis triukšmo lygis. Magnetiniai jutikliai gali būti panaudoti ir klasifikavimui, bet iš magnetinių parašų atrenkami požymiai nėra labai tikslūs, taigi kartu reikia naudoti požymius iš kitų jutiklių.

Pasinaudojant akselerometrų signalais galima gauti geriausių požymių lyginant su kitais sistemoje naudotais jutikliais. Pagal juos galima nustatyti TP ašių skaičių bei vietą signale, apskaičiuoti TP judėjimo greitį ir pataisyti iškreiptus magnetinius parašus. Jeigu sistemoje naudojami akselerometrai, greičio nustatymui nebereikia magnetinių jutiklių, be to, naudojant akselerometrus yra lengviau padidinti skaičiavimų tikslumą paliekant tą pačią skaičiavimų trukmę ir apimtį. Klasifikuoti pasinaudojant akselerometrų duomenimis galima tiesiogiai pagal ašių skaičių, taip pat, pasinaudojant atstumu tarp ašių galima klasifikuoti TP pagal atstumus tarp ašių bei atskirti 2-iejų ašių TP su priekabomis nuo 3-ijų ašių TP, tokių kaip sunkvežimis.

4. Klasifikavimas

Duomenų analizės metu TP buvo suskaidytos į 14 skirtingų klasių, kurių kiekvienoje buvo visos TP su vienodais kėbulo tipais. TP, važiuojančios su priekabomis, buvo atskirtos kaip atskira klasė (žr. 3 lentelę).

3 lentelė. Kiekvienos klasės pravažiavimų kiekiai

TP tipas	Kiekis
SUV	2990
Sedanai	2529
Universalai	1504
Hečbekai	1320
Maži furgonai	445
Furgonai	441
Vienatūriai	339
Maži sunkvežimiai 2 ašys	125
Pikapai	104
Sunkvežimiai 2 ašys	26
Sunkvežimiai 3 ašys	76
TP su priekabomis	159
Sunkvežimiai 4 ašys	62
Sunkvežimiai 5 ašys	16

Kaip galima matyti lentelėje, pravažiavusių skirtingų TP klasių kiekiai yra labai skirtingi – daugiausiai per duomenų rinkimo laikotarpį pravažiavo SUV klasės TP, o mažiausiai sunkvežimių su 5 ašimis. Atskiriant klases buvo siekiama jų sudaryti kuo daugiau tam, kad jas būtų galima sujungti į vieną klasę tuo atveju, jei šių klasių požymiai būtų per daug panašūs, kad jas pavyktų atskirti automatiškai pagal požymius. Taip pat, klasių atskyrimas smulkesnėmis dalimis padeda sumažinti duomenų triukšmą – atskyrus klases, jų požymių kitimo ribos taip pat atsiskiria ir dėl to pasidaro įmanoma atitinkamas klases atskirti nuo kitų. Tai pagerina klasifikavimo algoritmo tikslumą ir sumažina klasifikavimo modelio permokymo (angl. *overfitting*) tikimybę. Modelis būna permokytas tada, kai jis klasifikavimą su duomenimis, naudotais apmokymo metu, atlieka 100 % arba labai artimu 100 % tikslumu, bet naujus duomenis klasifikuoja netiksliai, kitaip tariant, modelis išmoksta duomenis, o ne sąryšius tarp jų.

Šiame skyriuje aprašomi ir palyginami rezultatai, gauti pasinaudojant skirtingais klasifikavimo metodais. TP yra skirstomos į klases, sudarytas remiantis automobilių kelių investicijų vadovu [17].

4.1. Sprendimų medis

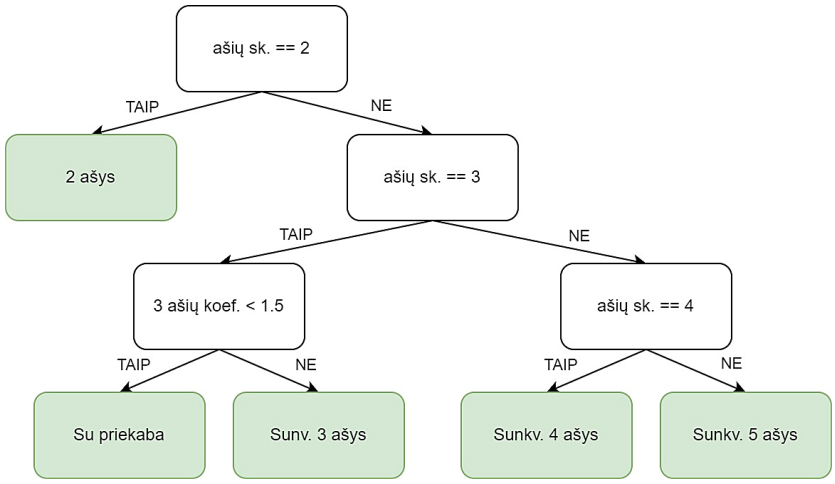
Sprendimų medžio algoritmas yra vienas populiariausių ir paprasčiausių dirbtinio intelekto modelių [18]. Šiuos medžius sudaro daug „jeigu" sąlygų, kurių kiekviena padalina duomenų rinkinį į 2 dalis, taip atskiriant skirtingas klases į skirtingas medžio šakas. Padalinus duomenų rinkinį pakankamai kartų taip, kad kiekvienoje pasirinkimų medžio šakoje būtų tik vienos klasės duomenys, šakojimas pasibaigia ir šis mazgas yra vadinamas medžio lapu. Idealiu atveju, jeigu klasifikavimo tikslumas yra

100 %, kiekviename iš šių medžio lapų yra tik vienos klasės duomenys. Kuo daugiau medžio lape yra duomenų iš kitų klasių, tuo klasifikavimo tikslumas yra mažesnis.

Prieš sudarant sprendimų medį reikia iš turimų duomenų atrinkti parametrus, kurie pakankamai gerai apibūdintų atitinkamą klasę. Keli tokių požymių pavyzdžiai – TP ašių kiekis arba atstumų tarp jų santykis (žr. 35 pav.). Pagal atrinktą požymį sudaromas mazgas kuriame suformuojama „jeigu" sąlyga su nustatyta riba, kuri geriausiai padalina duomenų rinkinį į 2 dalis. Požymiai, kuriais pasinaudojant galima padalinti duomenų rinkinį su didžiausiu tikslumu yra geriausi ir yra naudojami medžio pradžioje, o toliau naudojami vis prastesni požymiai tol, kol jie pagerina klasifikavimo rezultatus.

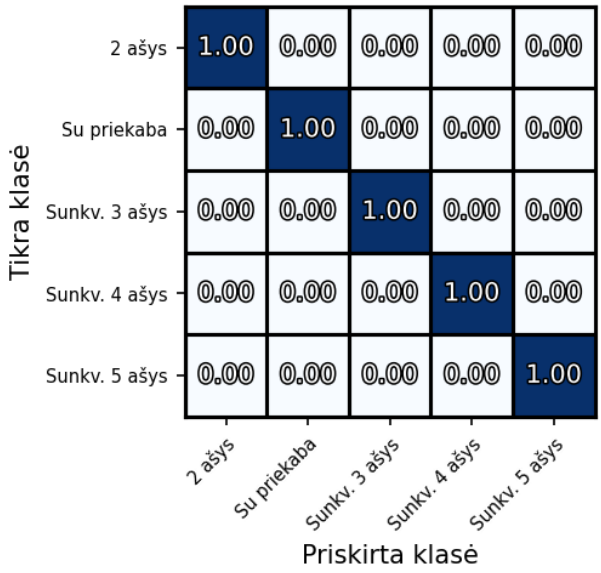
Sprendimų medį galima sudaryti statistiškai išanalizavus apskaičiuotus parametrus ir požymius ir tada pagal jų gerumą parenkant kuriuos iš jų naudoti anksčiau bei sudarant reikiamas „jeigu" sąlygas. Geriausi požymiai yra tie, kuriais pasinaudojus po padalinimo lapelyje lieka mažiausiai imčių iš kitų klasių. Alternatyva šiam būdui – automatinis medžio sudarymas. Požymių ieškojimui ir medžio sudarymui galima naudoti tam skirtas bibliotekas – tokiu būdu pagal nurodytus parametrus yra išanalizuojami funkcijai paduoti duomenys ir sudaromas sprendimų medžio modelis. Automatiniams medžio sudarymo metodams yra svarbu, kad modelio sudarymo funkcijai būtų paduotas vienodas visų klasių duomenų kiekis dėl to, kad medžio šakojimo metu yra žiūrimas rezultato mazgų užterštumas [18], [19]. Užterštumas nurodo kiek po padalinimo esančiuose lapeliuose yra neteisingai priskirtų klasių – sudarinėjant medį siekiama, kad visų lapelių užterštumas būtų lygus nuliui. Pavyzdžiui, jeigu duomenų rinkinyje yra 900 įrašų iš klasės A ir 100 įrašų iš klasės B, tai net visą duomenų rinkinį priskyrus klasei A užterštumas būtų nedidelis ir modelis galėtų suklasifikuoti tokį duomenų rinkinį su 90% tikslumu net ir nenaudodamas „jeigu" sąlygų. Šio tyrimo metu buvo surinkta daugiausiai duomenų iš SUV transporto priemonių, taigi sudarinėjant sprendimų medį ir norint šią klasę atskirti nuo, pavyzdžiui, mažų sunkvežimių su 2 ašimis, reikėtų SUV klasės pravažiavimų kiekį sumažinti iki 125 (žr. 3 lentelę) tam, kad būtų nagrinėjamas vienodas kiekis duomenų iš kiekvienos klasės. Taip pat galima padauginti mažiau duomenų turinčios klasės įrašus kopijuojant.

Be bibliotekų pagalbos sprendimų medį sudaryti galima pasinaudojant atrinktais požymiais, kurie gerai apibūdina atitinkamą klasę. Iš požymių, aprašytų 3 skyriuje, tam tinkamiausi yra požymiai, apskaičiuoti pasinaudojant akselerometrų duomenimis. Tokiu atveju klasifikavimas yra vykdomas tik pagal TP ašių skaičių ir atstumo tarp jų santykį – tokio medžio pavyzdys pateiktas 47 paveiksle.



47 pav. Paprastas klasifikavimo medis, sudarytas pasinaudojant akselerometro duomenimis

Kaip galima matyti, pasinaudojant tik ašių skaičiumi ir santykiu tarp jų, TP galima suklasifikuoti į 5 klases. Modelio klasifikavimo rezultatams įvertinti yra naudojamos klaidų matricos, kuriose atvaizduojamos tikimybės priskirti kiekvieną klasę vienai iš galimų klasių. Idealiu atveju visoje įstrižainėje yra tikimybės lygios vienetui – įstrižainės vidurkis nurodo bendrą modelio klasifikavimo tikslumą, taigi tokio modelio klasifikavimo tikslumas yra 100 %. Klasifikavimo medis (žr. 47 pav.), kuriuo gauta klaidų matrica pateikta 48 paveiksle, yra idealus – jo tikslumas yra 100 % dėl to, kad TP ašių kiekis yra unikalus požymis, kiekvienai klasei.



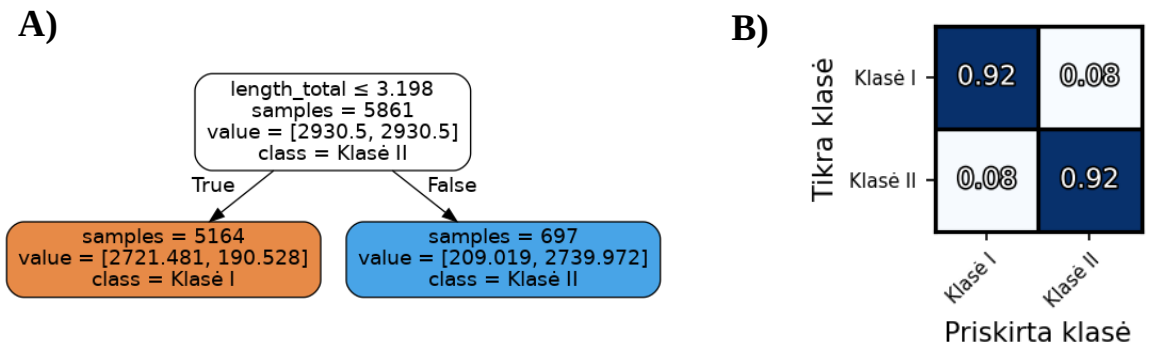
48 pav. Paprasto sprendimo medžio klasifikavimo rezultatų klaidų matrica

Klasifikavimui naudojant tokį sprendimų medį papildomų požymių nebereikia, nebent yra poreikis klasifikuoti į daugiau klasių. Tokiu atveju, reikia ieškoti daugiau požymių, kuriuos būtų galima panaudoti tam, kad atskirti papildomas TP klases. Vienas galimų požymių būtų TP ilgis. Tam, kad būtų galima nustatyti TP ilgį, reikalingas jos judėjimo greitis, kurį galima apskaičiuoti pasinaudojant magnetiniais jutikliais arba akselerometrais. Pastaruoju atveju ant kelio turi būti 2 grioveliai ar iškilimai, kaip aprašyta 3.2 skyriuje, o naudojant magnetinius jutiklius reikia, kad jie būtų du, išdėstyti tam tikru atstumu vienas nuo kito. Vienintelė klasė, kurią būtų galima padalinti smulkiau yra „2 ašys". Šiuo tikslu buvo atskirtos dvi didžiausios TP iš 2 ašių klasės – furgonai ir sunkvežimiai su 2 ašimis (žr. 4 lentelę).

4 lentelė. TP tipai priskirti atitinkamoms klasėms klasifikavimui į 2 klases

TP tipas	**Klasė**
SUV, sedanai, universalai, hečbekai, maži furgonai, vienatūriai, maži sunkvežimiai su 2 ašimis, pikapai	I
Furgonai, sunkvežimiai su 2 ašimis	II

Tam, kad atskirti šias dvi klases pakanka vieno požymio – apskaičiuoto TP ilgio. Šiuo atveju nustatomas slenkstis 3,2 m ir tada visos 2 ašių TP, kurių ilgis yra mažesnis priskiriamos I klasei, o visos kitos – II klasei. Šis klasifikavimo medis ir jo klasifikavimo klaidų matrica pateikta 49 paveiksle.



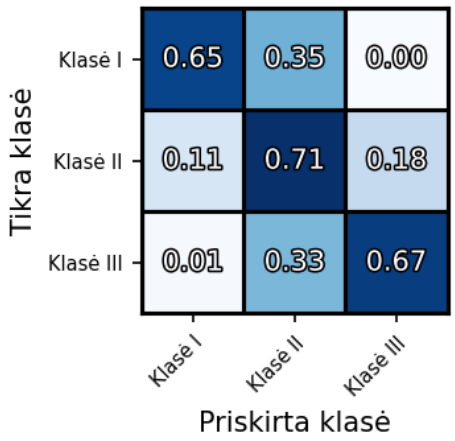
49 pav. TP su 2 ašimis klasifikavimo į 2 smulkesnes klases pasirinkimų medis (A) ir klaidų matrica (B)

Kaip galima matyti, tokio medžio klasifikavimo tikslumas yra 92 %. Kadangi pirmoje klasėje yra gerokai daugiau TP tipų, šią klasę galima padalinti dar kartą. Likusios didžiausios transporto priemonės yra mažesni furgonai, pikapai ir mažesni sunkvežimiai su 2 ašimis (žr. 5 lentelę).

5 lentelė. TP tipai priskirti atitinkamoms klasėms klasifikavimui į 3 klases

TP tipas	**Klasė**
SUV, sedanai, universalai, hečbekai, vienatūriai	I
Maži sunkvežimiai su 2 ašimis, pikapai, maži furgonai	II
Furgonai, sunkvežimiai su 2 ašimis	III

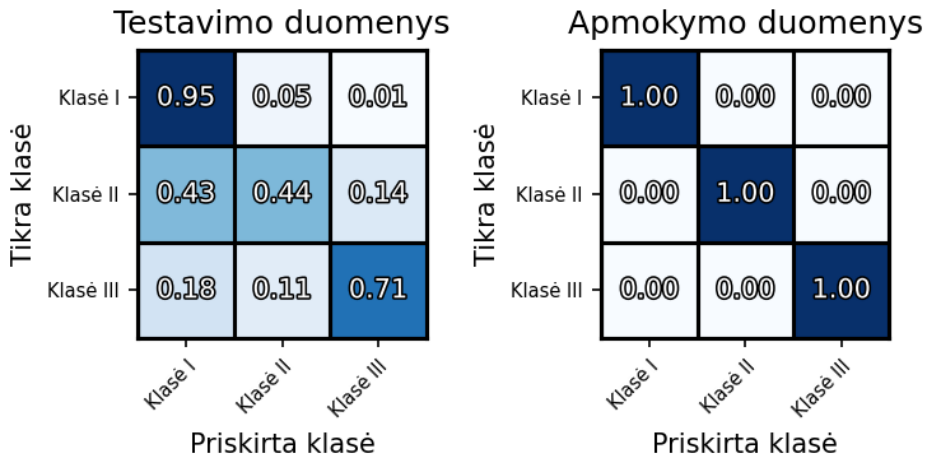
Transporto priemonių su 2 ašimis atskyrimui į 3 smulkesnes klases naudojant tik ilgį, gaunami rezultatai yra gerokai prastesni nei dalinant į 2 klases – tokio modelio klasifikavimo tikslumas yra tik 68 % (žr. 50 pav.).



50 pav. TP su 2 ašimis klasifikavimo į 2 smulkesnes klases klasifikavimo medis (A) ir klaidų matrica (B)

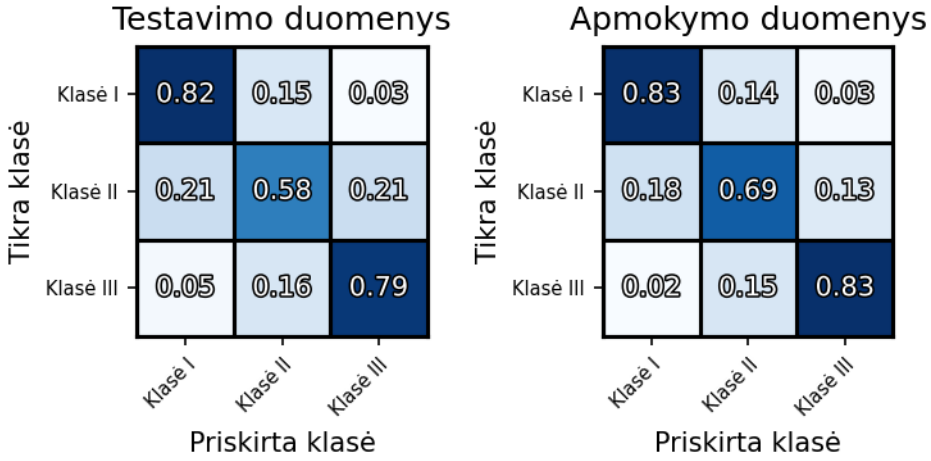
Pagerinti modelio klasifikavimo tikslumą galima pasinaudojant daugiau požymių. Klasifikavimo medžio kūrimo funkcija gali atrinkti naudingiausius požymius ir panaudoti tik juos, taigi jai galima paduoti visus 72 surinktus požymius (žr. 1 priedą). Šis požymių skaičius yra labai didelis, ypač atsižvelgiant į tai, kad kai kurių klasių pravažiavimų skaičius yra nedaug didesnis (pavyzdžiui, buvo

užfiksuoti tik 24 sunkvežimių su 2 ašimis pravažiavimų duomenys). Dėl šios priežasties gali atsitikti taip, kad modelis bus permokytas. Permokytas modelis pasižymi tuo, kad jo klasifikavimo tikslumas yra 100 %, o pats klasifikavimo medis turi labai daug lapelių, kuriuose yra atskirtos vos kelios imtys. Vienas būdų aptikti persimokymo reiškiniui yra modelio apmokymui ir testavimui naudoti du atskirus duomenų rinkinius – testavimo ir apmokymo. Jeigu modelis yra permokytas, tikslumas su apmokymo duomenimis bus labai aukštas, o su testavimo duomenimis – gerokai prastesnis. Permokyto pasirinkimų medžio, sudaryto pasinaudojant visais atrinktais požymiais, klaidų matricos pateiktos 51 paveiksle.



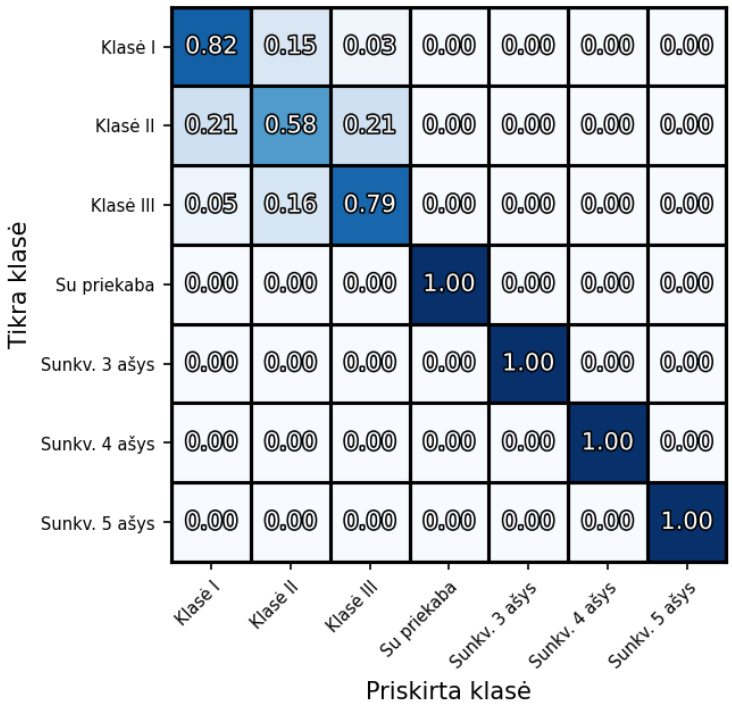
51 pav. Klaidų matricos su testavimo ir apmokymo duomenimis gautos panaudojus permokytą modelį

Kaip galima matyti, klaidų matrica sudaryta naudojantis apmokymo duomenimis yra tobula, o naudojant testavimo duomenis, kurių modelio sudarymo funkcija nėra mačiusi, rezultatai gerokai prastesni – II klasės klasifikavimo tikslumas siekia 44 %, o bendras tokio medžio tikslumas – 69 %. Vienas būdų šiai problemai spręsti yra paderinti modelio kūrimo funkcijai paduodamus parametrus, tokius kaip maksimalus galimas lapelių skaičius, medžio gylis arba modeliui naudojamų požymių skaičius [20]. Šiuo atveju, apribojus lapų skaičių iki 7, maksimalų gylį iki 5, mažiausią lapo imčių kiekį iki 30 ir mažiausią reikiamą imčių kiekį prieš dalinimą iki 60 buvo išvengta permokymo reiškinio (žr. 52 pav.).



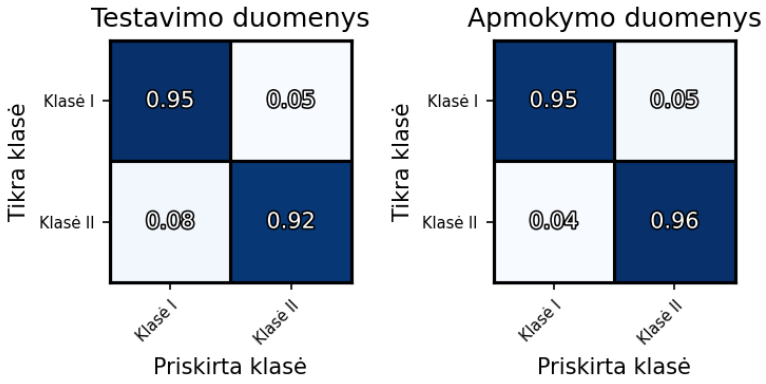
52 pav. Nepermokyto modelio klaidų matricos

Kaip galima matyti, matricos su apmokymo ir testavimo duomenimis yra panašios – tai reiškia, kad modelis buvo gerai pritaikytas klasifikavimo uždaviniui spręsti ir nebuvo permokytas, priešingai nei 51 paveikslo atveju. Pasinaudojant tokiu sprendimų medžiu, klasifikavimą galima atlikti su 73 % tikslumu. Šiam modeliui medžio sudarymo funkcija atrinko 11 požymių: TP ilgis, magnetinio parašo ir X ašies signalo pirmos eilės išvestinės verčių standartinė deviacija, Z ašies signalo didžiausia ir mažiausia vertės, normuoto Z ašies signalo ir magnetinio parašo 5 dalies vidurkis, nenormuoto Z ašies signalo ir magnetinio parašo 3 dalies vidurkis ir X ašies signalo 2 dalies vidurkis ir galiausiai X ašies signalo, suvienodinto iki 300 taškų, išvestinės verčių suma. Gautas rezultatas panašus ar net geresnis lyginant su kai kuriais literatūros šaltiniais, kuriuose buvo skirstoma į panašias klases pasinaudojant tik magnetiniais jutikliais (65 % [8], 63 % [21], neskaičiuojant motociklų 76 % [22]). Sujungus 48 ir 52 paveikslų modelius, bendras medžio tikslumas būtų 88 % klasifikuojant TP į 7 klases (žr. 53 pav.). Šiam modeliui būtų reikalingi požymiai, apskaičiuoti pagal akselerometro ir magnetinių jutiklių signalus.



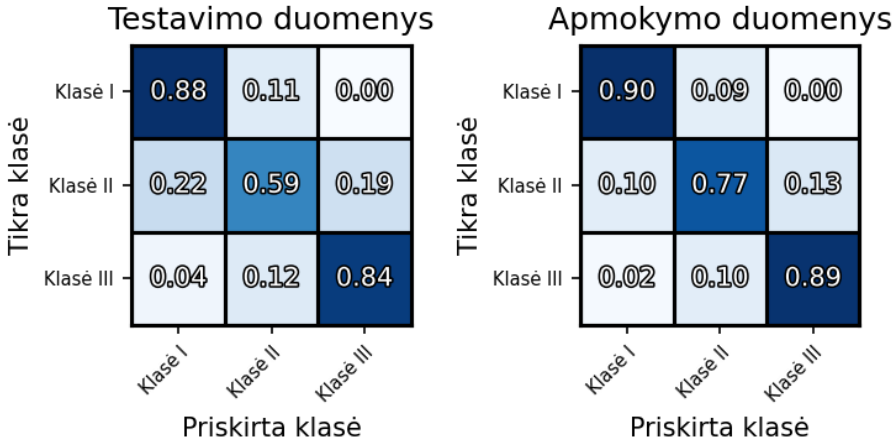
53 pav. Klaidų matrica naudojant bendrą akselerometrų ir magnetinių jutiklių požymių medį

Dar vienas būdas sumažinti permokymo tikimybę ir tuo pačiu pagerinti klasifikavimo tikslumą yra naudoti atsitiktinį mišką (angl. *random forest*) [20]. Atsitiktinis miškas yra pasirinkimų medžių rinkinys, kurių kiekvienas yra sudaromas ne iš visų paduodamų duomenų, o iš jų atrenkant atsitiktinius požymių ir imčių rinkinius. Tokio modelio naudojimo metu visi jame esantys medžiai atlieka klasifikavimą atskirai, o pabaigoje vykdomas balsavimas ir galutinis sprendimas priimamas pagal balsų skaičių arba pagal tikimybių, kurias atskirai priskiria kiekvienas medis, vidurkį [23] [24]. Klaidų matricos, gautos pasinaudojant atsitiktinio miško modeliu klasifikuojant dviejų ašių TP į 2 ir 3 klases, pavaizduotos atitinkamai 54 ir 55 paveiksluose.



54 pav. Klaidų matricos, gautos klasifikuojant į 2 klases atsitiktinio miško modeliu

Atsitiktinio miško modeliu klasifikuojant į 2 klases gaunamas 2,5 % didesnis tikslumas nei klasifikuojant vienu pasirinkimų medžiu. Didžiausių transporto priemonių klasės tikslumas lieka toks pats, bet I klasė yra atskiriama 3 % tiksliau.



55 pav. Klaidų matricos, gautos klasifikuojant į 3 klases atsitiktinio miško modeliu

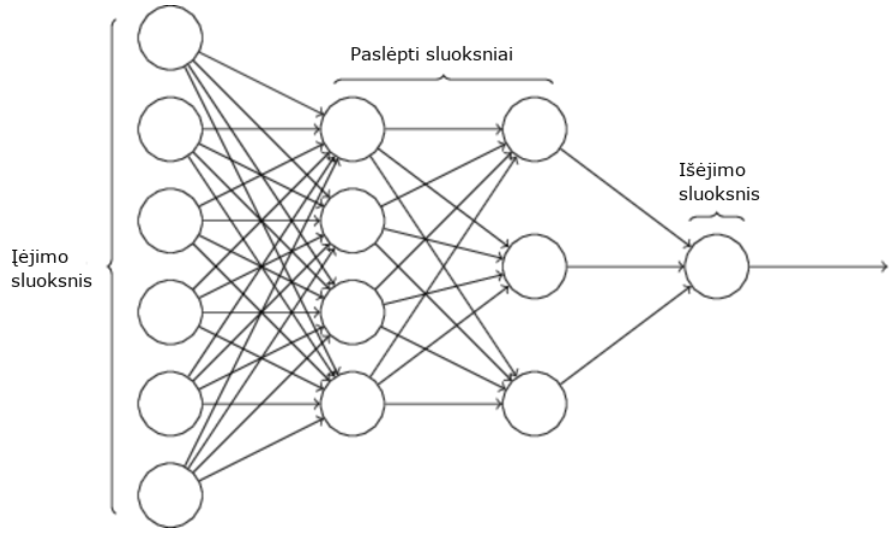
Dviejų ašių TP klasifikuojant į 3 klases atsitiktinio miško modeliu buvo gautas 77 % tikslumas, taigi, pasinaudojant šiuo metodu buvo gautas 4 % didesnis tikslumas nei naudojant vieno medžio modelį. Kaip galima matyti klaidų matricoje, naudojant atsitiktinį mišką, taip pat kaip ir vieno medžio atveju, modeliui yra sunku atskirti II klasę. Vis dėlto, modelis I klasę klasifikuoja 6 %, II klasę – 1 %, o III klasę – 5 % tiksliau. Taip yra dėl to, kad, lyginant su paprastu pasirinkimų medžiu, atsitiktinis miškas yra mažiau jautrus klaidoms ir kraštutinėms išsišokančioms reikšmėms, kurių pasitaiko požymiuose, nes atsitiktinio miško metodo pabaigoje visų medžių rezultatai yra vidurkinami taip sumažinant išsišokančių reikšmių įtaką. Taip pat kaip ir vieno medžio atveju, šį modelį galima sujungti su modeliu, skirtu klasifikuoti TP į 2 klases pasinaudojant iš akselerometrų gautais požymiais. Šiuo atveju bendras modelio tikslumas būtų 90 %.

Nors pasirinkimų medis yra vienas paprasčiausių dirbtinio intelekto modelių, jį kuriant yra galimybė pasirinkti įvairių parametrų, kurie turi įtakos rezultatams, taip pat, sudėtingesnėms užduotims galima pasinaudoti keliais pasirinkimų medžiais vienu metu – atsitiktiniu mišku. Didžiausias klasifikavimo tikslumas gaunamas transporto priemones klasifikuojant į 5 klases pasinaudojant ašių skaičiumi,

nustatytu pagal akselerometrų duomenimis. Vis dėlto, šis modelis netinka TP su 2 ašimis klasifikavimui – tokiu atveju reikia pasinaudoti kitais apskaičiuotais požymiais iš magnetinių jutiklių.

4.2. Neuroniniai tinklai

Neuroniniai tinklai yra dirbtinio intelekto modeliai, paremti smegenų struktūra ir veikimo principu. Juos sudaro sluoksniais sujungti mazgai, vadinami neuronais. Neuronai būna išdėstyti 3 rūšių sluoksniais: įėjimo, išėjimo ir paslėptais (žr. 56 pav.). [25] [26]



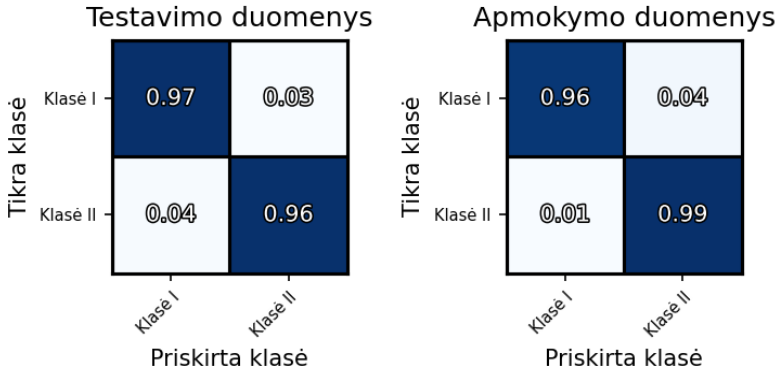
56 pav. Neuroninio tinklo struktūra [25]

Tiesiaeigiame (angl. *feedforward*) tinkle neuronai būna sujungiami kryptimi nuo įėjimo sluoksnio link išėjimo sluoksnio, t.y. pirmojo paslėpto sluoksnio neuronų reikšmės priklauso nuo įėjimo sluoksnio reikšmių, antrojo sluoksnio - nuo pirmojo sluoksnio ir t.t., kol išėjimo sluoksnyje yra gaunamas rezultatas. Kiekvienas sujungimas tarp neuronų turi savo svorį, kuris yra sudauginamas su ankstesnio neurono išėjimu. Neuronuose yra susumuojamos ankstesnių neuronų ir atitinkamų svorių sandaugos ir gaunamas rezultatas yra to neurono vertė, kuri toliau perduodama į tolimesnius neuronus [27]. Sujungimų svoriai yra derinami dirbtinio intelekto modelio apmokymo metu. Priklausomai nuo neuronų ir paslėptų sluoksnių skaičiaus priklauso modelio sudėtingumas ir greitaveika.

Kaip ir sprendimų medis, neuroninis tinklas gali būti permokytas. Tam, kad būtų išvengta šio reiškinio, yra naudojami keli skirtingi metodai, kuriuos, esant poreikiui, galima taikyti ir vienu metu. Vienas paprasčiausių būdų išvengti permokymo reiškinio yra ankstyvas sustabdymas. Pastebėjus permokymo požymių, pavyzdžiui, kad modelis labai gerai klasifikuoja apmokymo duomenis, bet palyginus gerokai prasčiau klasifikuoja testavimo duomenis, galima sumažinti modelio apmokymo iteracijų (epochų) skaičių, taip užkertant kelią modeliui išmokti jam paduotą duomenų rinkinį. Kitas naudojamas būdas – išmetimo sluoksnis. Jo paskirtis – apmokymo metu kiekvienoje epochoje išjungti nurodytą dalį atsitiktinių ankstesnio sluoksnio neuronų taip sumažinant modelio sudėtingumą ir priverčiant modelį ieškoti kitų naudingų neuronų, požymių ir priklausomybių problemai spręsti. Dar vienas būdas naudojamų požymių atrinkimui yra L1 reguliavimas vadinamas LASSO. Šis metodas įveda nuostolius jeigu modelis naudoja per daug požymių. Tokiu būdu modelis yra skatinamas sumažinti mažai naudingų požymių svorius iki 0, paliekant tik naudingiausius požymius. Kitas populiarus reguliavimo metodas turintis įtakos modelio svoriams yra svorių mažinimo, žymimas kaip L2. Šis metodas skiriasi nuo L1 tuo, kad jis skatina modelį palaikyti svorius mažus ir panašaus dydžio.

Taip modeliui neleidžiama susikoncentruoti tik į kelis požymius, kurie gali atrodyti svarbūs apmokymo duomenų rinkinyje, bet iš tikrųjų būti ne tokie svarbūs visuose kituose duomenyse. [28] [29]

Sprendimų medžiai yra paprastesni ir veikia greičiau negu neuroniniai tinklai, bet neuroniniai tinklai gali išmokti atpažinti sudėtingesnes priklausomybes. Pasinaudojant pasirinkimų medžiais, TP buvo suklasifikuotos pagal jų ašių skaičių į 5 klases su 100 % tikimybe, taigi šio tyrimo metu neuroniniais tinklais buvo siekiama pagerinti rezultatus klasifikuojant tik TP su 2 ašimis. Pirmiausia buvo sukurtas neuroninio tinklo modelis, skirtas klasifikuoti TP į 2 klases, dėl to, kad pasinaudojant pasirinkimų medžiais klasifikuojant į 2 klases buvo pasiektas didžiausias tikslumas. Šio modelio klaidų matrica klasifikuojant tas pačias klases (žr. 4 lentelę) pateikta 57 paveiksle.



57 pav. Klasifikavimo į 2 klases, pasinaudojant neuroniniu tinklu, klaidų matricos

Kaip galima matyti paveiksle, šis modelis klasifikavimą atlieka su 96 % tikslumu ir abi TP klases atskiria geriau negu ankstesni modeliai. Toliau buvo sukurtas modelis, skirtas klasifikuoti TP į 3 klases (žr. 5 lentelę). Šio modelio struktūra pateikta 58 paveiksle.

InputLayer		→	Dense		→	Dropout		→	Dense		→	Dense	
input:	output:		input:	output:		input:	output:		input:	output:		input:	output:
[(None, 72)]	[(None, 72)]		(None, 72)	(None, 73)		(None, 73)	(None, 73)		(None, 73)	(None, 50)		(None, 50)	(None, 3)

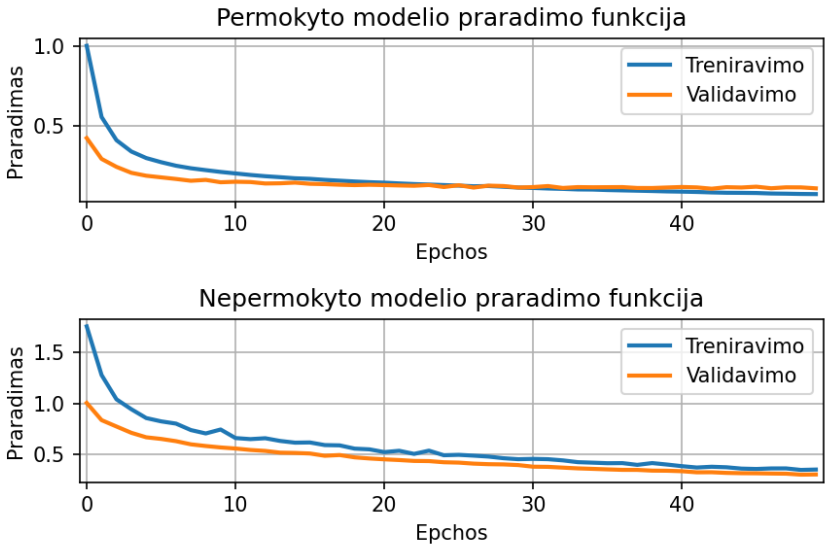
58 pav. Neuroninio tinklo, skirto klasifikuoti 3 tipų TP, struktūra

Pirmasis modelio sluoksnis yra įėjimo sluoksnis ir jis turi 72 neuronus dėl to, kad iš kiekvieno signalo yra atrenkama po 72 požymius. Antrasis sluoksnis, kuris tuo pačiu yra pirmasis paslėptas sluoksnis, turi tiek pat įėjimų kiek ankstesnis sluoksnis išėjimų, t.y. 72, o išėjimų turi vienu daugiau. Internete galima rasti rekomendacijų sluoksnio neuronų skaičiaus nustatymui, bet tai yra tik rekomendacijos, duodamos kaip pradinis taškas nuo kurio galima pradėti bandymus ieškant optimalaus neuronų skaičiaus [30]. Šio ir toliau aprašomų sluoksnių neuronų kiekiai buvo surasti eksperimentiškai derinant parametrus taip, kad būtų gauti geriausi rezultatai. Po pirmojo paslėpto sluoksnio eina išmetimo (angl. *dropout*) sluoksnis, toliau - antrasis paslėptas sluoksnis ir galiausiai išėjimo sluoksnis, kuris turi 3 neuronus – kiekvienai klasei po vieną.

Pirmajame paslėptame modelio sluoksnyje buvo naudojamas L1 reguliavimas, o po jo – išmetimo sluoksnis. Abi šios priemonės skirtos sumažinti naudojamų požymių kiekį ir tuo pačiu sumažinti permokymo tikimybę. Tai yra naudinga dėl to, kad tarp apskaičiuotų požymių yra tokių, kurie yra

mažiau naudingi nei kiti, taigi pasinaudojant L1 reguliavimu ir išmetimo sluoksniu modelis gali atsirinkti kuriuos iš požymių naudoti. Antrame paslėptame sluoksnyje buvo naudojamas L2 reguliavimas, taip buvo pasiekta, kad modelis neuždėtų per didelių svorių keliems iš anksčiau atrinktų požymių ir bandytų panaudoti kuo daugiau iš jų.

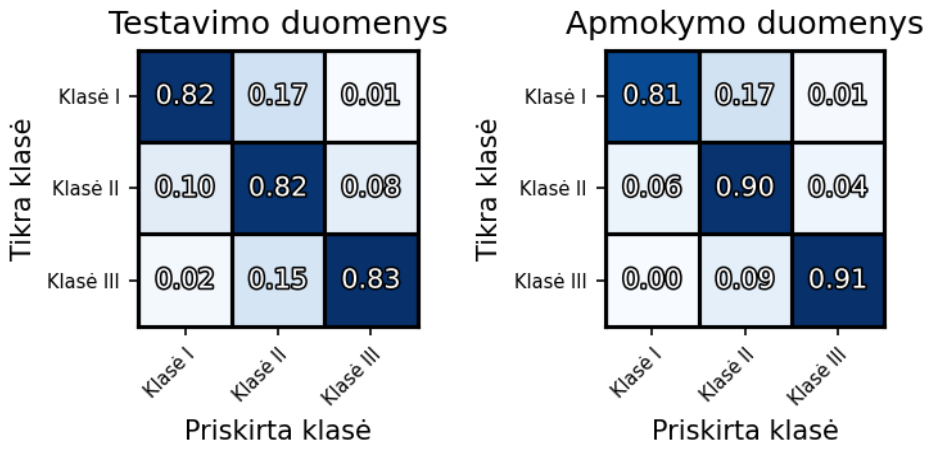
Nenaudojant reguliatorių, išmetimo sluoksnių, mokymo epochų ribojimo ar kitų permokymo išvengimo priemonių modelis yra permokomas. Vienas būdų tai pamatyti yra stebėti modelio mokymo metu vertintos praradimo funkcijos kitimo grafiką (žr. 59 pav.). Dėl šios priežasties modelio apmokymui reikia naudoti treniravimo ir validavimo duomenų rinkinius. Treniravimo duomenys yra naudojami modelio treniravimui, o validavimo – modelio patikrinimui treniravimo metu po kiekvienos epochos. Taip pat yra reikalingas testavimo duomenų rinkinys tam, kad modelį būtų galima patikrinti po apmokymo su duomenimis, kurių modelis nebuvo matęs treniravimo metu. Modeliams apmokyti šio tyrimo metu visi turimi duomenys (viso apie 10 tūkst. imčių) buvo padalinti į 3 dalis: 50 % - treniravimui, 20 % - validavimui ir likę 30 % - testavimui. Modelio apmokymo metu yra stebimas modelio praradimas – tai yra matas be dimensijos, kurį modelis apmokymo metu stengiasi privesti iki 0. 59 paveiksle pateikti jo kitimo istorijos pavyzdžiai kai modelis yra permokomas ir nepermokomas.



59 pav. Neuroninio tinklo praradimo istorija kai modelis yra permokomas ir nepermokomas

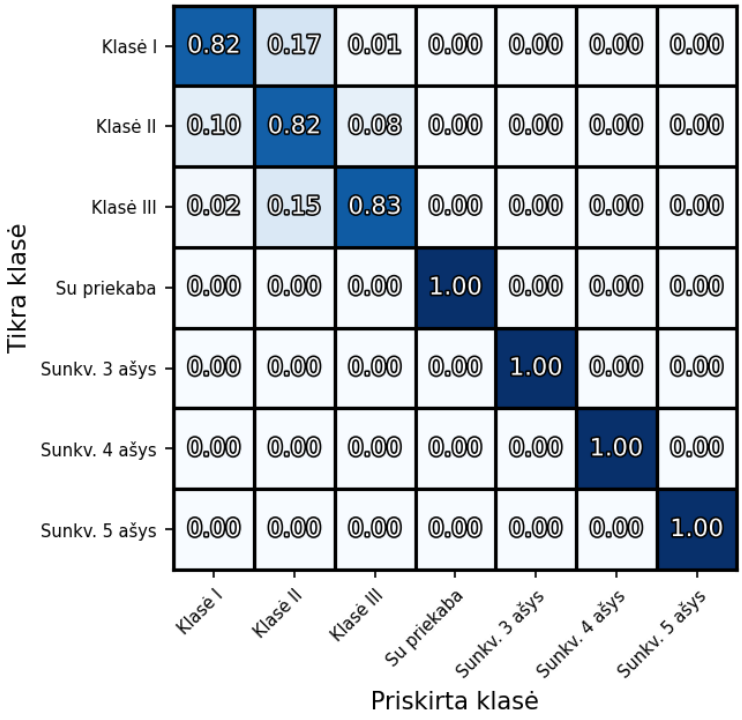
Permokyto modelio atveju vykdant apmokymą ties 20 epocha treniravimo paklaida mažėjo, bet validavimo paklaida jau pradėjo didėti – tai yra vienas požymių, kad prasideda permokymo procesas. Šio reiškinio galima išvengti pritaikant anksčiau minėtus metodus – apribojant epochų kiekį, pasinaudojant L1 ir L2 reguliatoriais bei panaudojant išmetimo sluoksnius. Pritaikius šiuos metodus yra išvengiama permokymo reiškinio net apmokant modelį ilgiau nei anksčiau, bet tuo pačiu pagerėja ir klasifikavimo tikslumas dėl to, kad modelis gali toliau treniruotis naudodamas jam paduotą treniravimo duomenų rinkinį – permokymo atveju, jis pasiekia 100 % tikslumą paduotam rinkiniui ir nebegali toliau tobulėti, nes modelis „įsiminė“ duomenų rinkinio vertes.

Nepermokyto modelio, skirto klasifikuotį į 3 klases klaidų matricos pateiktos 60 paveiksle. Šiuo modeliu buvo gautas 82 % tikslumas.



60 pav. Klasifikavimo į 3 klases, pasinaudojant neuroniniais tinklais, klaidų matrica

Tai, kad modelis buvo nepermokytas matosi 60 paveiksle – klasifikavimo rezultatai su apmokymo duomenimis nesiekia 100 % tikslumo, be to, klasifikavimo rezultatai yra panašūs ir apmokant modelį ir jį testuojant. Lyginant su ankstesniais modeliais, I ir III klasių klasifikavimo tikslumas yra šiek tiek mažesnis, bet II klasės klasifikavimo tikslumas yra net 23 % didesnis. Naudojant šį modelį kartu su klasifikavimo medžiu, kuris klasifikuoja TP pagal jų ašių skaičių, gaunama 61 paveiksle pavaizduota klaidų matrica.



61 pav. Klaidų matrica gauta panaudojus du modelius

Kaip galima matyti, I, II ir III klasių klasifikavimo tikslumas yra mažesnis negu likusių klasių. Bendras modelio tikslumas gaunamas 92 %. Tikslumą galima padidinti vietoj neuroninio tinklo, kuris klasifikuoja į 3 klases, naudojant neuroninį tinklą, klasifikuojantį į 2 klases. Tokiu atveju modelis

nebegalėtų atpažinti II klasės TP (mažų furgonų ir sunkvežimių), bet klasifikavimą atliktų su 99% tikslumu.

4.3. Klasifikavimo rezultatų apibendrinimas

Pasinaudojant požymiais, apskaičiuotais iš signalų, užfiksuotų pravažiuojant įvairioms TP virš jutiklių bloko, klasifikavimą galima atlikti keliais skirtingais būdais. Visų anksčiau aprašytų modelių rezultatai pateikti 6 lentelėje.

6 lentelė. Tyrimų rezultatai ir klasifikavimo modelių palyginimas

Modelio tipas	**Klasių skaičius**	**Tikslumas, %**	**Reikalingi jutikliai**
Pasirinkimų medis	5	100	Akselerometras
Pasirinkimų medis	6	97	Akselerometras ir magnetinis jutiklis
Atsitiktinis miškas	6	98	
Neuroninis tinklas	6	99	
Pasirinkimų medis	7	88	
Atsitiktinis miškas	7	90	
Neuroninis tinklas	7	92	

Priklausomai nuo poreikių, tiksliausiai klasifikuojantys modeliai gali būti skirtingi. Jeigu reikia TP suklasifikuoti į kelias stambias klases, galima pasinaudoti paprastu sprendimų medžiu kuris atliktų klasifikavimą 100 % tikslumu į 5 klases. Šiam modeliui pakanka tik akselerometro. Vis dėlto, šis modelis gali klasifikuoti tik pagal apskaičiuotą TP ašių kiekį, pavyzdžiui, sedaną priskirtų tai pačiai klasei kaip sunkvežimį su dviem ašimis. Jeigu reikia atskirti šias dvi klases reikalingas papildomas magnetinis jutiklis. Tokiu atveju pirmiausia būtų panaudojamas modelis, klasifikuojantis pagal TP ašių skaičių, o tada 2 ašių TP būtų suklasifikuojamos dar smulkiau pasinaudojant neuroninio tinklo modeliu. Šis neuroninis tinklas gali iš 2 ašių TP aibės atskirti sunkvežimius ir furgonus, gaunant 99 % jungtinio modelio tikslumą. Norint papildomai atskirti ir mažus furgonus bei sunkvežimius modelio tikslumas sumažėja iki 92 %, bet tada galima atlikti klasifikavimą į 7 klases.

Pasinaudojant galutiniu modeliu gaunama 4 klasėmis mažiau negu aprašoma automobilių kelių investicijų vadove [17]. Maži autobusai buvo sujungti į vieną klasę kartu su mažais ir vidutiniais sunkvežimiais su 2 ašimis dėl panašių požymių. Taip pat buvo atskirai klasifikuojamos transporto priemonės su priekabomis, o motociklų, didelių autobusų ir traktorių kelyje, kur buvo įdiegta duomenų surinkimo sistema, visai nebuvo užfiksuota. Motociklus atskirti nuo kitų klasių būtų galima pagal mažesnį atstumą tarp ašių, o didelį autobusą – pagal didesnį atstumą tarp ašių. Traktorius nuo kitų klasių galėtų atsiskirti savo magnetiniu parašu, nes jo kėbulo forma labai skiriasi nuo kitų transporto priemonių.

Išvados

1. Transporto priemonių klasifikavimo uždaviniui spręsti dažniausiai naudojamos vaizdo kameros, nes jomis pasiekiamas didžiausias klasių kiekis ir vienas didžiausių klasifikavimo tikslumų. Dėl trūkumų, tokių kaip suprastėjęs klasifikavimo tikslumas esant prastam matomumui per lietų ar rūką, egzistuoja ir alternatyvūs sprendimai. Šių trūkumų galima išvengti naudojant kitus jutiklius, pavyzdžiui, pjezoelektrinius elementus ar magnetinius jutiklius. Lyginant TP klasifikavimo tyrimų duomenis, geresnius rezultatus straipsnių autoriai pasiekdavo naudodami DI modelius.
2. Išanalizavus MEMS ir *fluxgate* magnetinių jutiklių duomenis buvo nustatyta, kad TP magnetinių parašų analizei pakankamas jutiklių diskretizavimo dažnis yra 100 Hz, dėl to, kad aukštesniuose dažniuose nėra jokios papildomos informacijos. Klasifikavimo uždaviniui labiau tinkamas yra *fluxgate* tipo magnetinis jutiklis dėl mažesnio triukšmo signale. Skaičiuojant TP greitį didesnis tikslumas gaunamas naudojant didesnį diskretizavimo dažnį, taigi šioje sistemoje greičio skaičiavimui tam labiau tinka MEMS tipo jutikliai dėl dvigubai didesnio diskretizavimo dažnio.
3. Akselerometrų signalas suteikia įvairios informacijos apie pravažiavusią TP:
    - signale galima aptikti ir suskaičiuoti TP ašių skaičių;
    - pagal akselerometro signalą galima apskaičiuoti TP judėjimo greitį – tai yra alternatyva greičio nustatymo, pagal magnetinius parašus, metodui;
    - aptikus TP ašis signale ir apskaičiavus TP judėjimo greitį galima nustatyti ir TP ilgį.
4. Mikrofonai ir pjezoelektrinės gembės tinka vibracijų matavimui, bet sistemai, kurioje jau yra akselerometras, neduoda jokios papildomos informacijos. Sistemoje be akselerometrų, daugumą metodų, susijusių su akselerometrų signalais, būtų galima pritaikyti ir mikrofonų ar pjezoelektrinių gembių signalams, bet rezultatai būtų prastesni. Pjezoelektrinių gembių atveju taip yra dėl to, kad po vibracijų sužadinimo jos ilgai nenuslopsta, o mikrofonų atveju – dėl didelio signalo triukšmo ir dėl nenuspėjamo pikų atsiradimo TP važiuojant per griovelius.
5. Pasinaudojant TP ašių skaičiumi ir atstumų tarp ašių santykiu, apskaičiuotais pagal akselerometrų duomenis, visi 10136 turimi transporto priemonių įrašai buvo teisingai suskirstyti į 5 klases. Sprendimų medis, sudarytas spręsti šiai užduočiai negali atskirti 2 ašių transporto priemonių, priklausančių skirtingoms klasėms, tokių kaip lengvasis automobilis ir didelis sunkvežimis. Norint tai atlikti reikia pasinaudoti papildomais požymiais, apskaičiuotais iš magnetinių jutiklių.
6. Lyginant sprendimų medį, atsitiktinį mišką ir neuroninį tinklą didžiausias klasifikavimo tikslumas gaunamas naudojant neuroninį tinklą. Šį modelį sujungus su modeliu, klasifikuojančiu į 5 klases pagal TP ašis, gaunamas bendras modelis, kuriuo galima pasiekti 99 % tikslumą klasifikuojant į 6 klases ir 92 % tikslumą klasifikuojant į 7 klases.

Literatūros sąrašas

[1] M. Biglari, A. Soleimani, H. Hassanpour, „A Cascaded Part-Based System for Fine-Grained Vehicle Classification," *IEEE Transactions on Intelligent Transportation Systems,* t. 19, pp. 273-283, 2018.

[2] M. A. Hedeya, A. H. Eid, R. F. Abdel-Kader, „A Super-Learner Ensemble of Deep Networks for Vehicle-Type Classification," *IEEE Access,* t. 8, pp. 98266-98280, 2020.

[3] Y. Chen, W. Hu, „A Video-Based Method With Strong-Robustness for Vehicle Detection and Classification Based on Static Appearance Features and Motion Features," *IEEE Access,* t. 9, pp. 13083-13098, 2021.

[4] A. Ramanathan, M. Chen, „Spatiotemporal Vehicle Tracking, Counting and Classification," įtraukta *IEEE Third International Conference on Multimedia Big Data*, 2017.

[5] Zhiming Luo , Frédéric Branchaud-Charron, Carl Lemaire, Janusz Konrad, Shaozi Li, Akshaya Mishra, Andrew Achkar, Justin Eichel, Pierre-Marc Jodoin, „MIO-TCD: A New Benchmark Dataset for Vehicle Classification and Localization," *IEEE TRANSACTIONS ON IMAGE PROCESSING, vol. 27,* pp. 5129-5141, 2018.

[6] G. Burresi, R. Giorgi, „A field experience for a vehicle recognition system using magnetic sensors," *4th Mediterranean Conference on Embedded Computing (MECO),* pp. 178-181, 2015.

[7] B. Yang, Y. Lei, „Vehicle Detection and Classification for Low-Speed Congested Traffic With Anisotropic Magnetoresistive Sensor," *IEEE Sensors Journal,* t. 15, nr. 2, pp. 1132-1138, 2015.

[8] X. Chen, X. Kong, M. Xu, K. Sandrasegaran, J. Zheng, „Road Vehicle Detection and Classification Using Magnetic Field Measurement," *IEEE Access,* t. 7, pp. 52622-52633, 2019.

[9] W. Li, Z. Liu, Y. Hui, L. Yang, R. Chen and X. Xiao, „Vehicle Classification and Speed Estimation Based on a Single Magnetic Sensor," *IEEE Access,* t. 8, pp. 126814-126824, 2020.

[10] H. Liu, J. Ma, T. Xu, W. Yan, L. Ma, X. Zhang, „Vehicle Detection and Classification Using Distributed Fiber Optic Acoustic Sensing," *IEEE Transactions on Vehicular Technology,* t. 69, nr. 2, pp. 1363-1374, 2020.

[11] B. Dawton, „Initial Evaluation of Vehicle Type Identification using Roadside Stereo Microphones," *IEEE Sensors Applications Symposium (SAS),* pp. pp. 1-6, 2020.

[12] E. Odat, J. S. Shamma, C. Claudel, „Vehicle Classification and Speed Estimation Using Combined Passive Infrared/Ultrasonic Sensors," *IEEE Transactions on Intelligent Transportation Systems,* t. 19, nr. 5, pp. 1593-1606, 2018.

[13] B. Sliwa, N. Piatkowski, C. Wietfeld, „The Channel as a Traffic Sensor: Vehicle Detection and Classification Based on Radio Fingerprinting," *IEEE Internet of Things Journal,* t. 7, nr. 8, pp. 7392-7406, 2020.

[14] S. A. Rajab and H. H. Refai, „A single element piezoelectric sensor for vehicle classification using the ratio of track width to length," *IEEE Intelligent Vehicles Symposium Proceedings,* pp. pp. 1463-1468, 2014.

[15] G. Stevens, „Average Car Length: All You Need to Know About It,“ [Tinkle]. Prieiga internete: https://www.way.com/blog/average-car-length/.

[16] M. Ambraziūnas, V. Markevičius, D. Navikas, D. Miklušis, J. Balamutas and D. Andriukaitis, „Application of Accelerometer for Vehicle Parameterization,“ įtraukta *XXXI International Scientific Conference Electronics (ET),* Sozopol, 2022.

[17] „Dėl Automobilių kelių investicijų vadovo patvirtinimo (Automobilių kelių investicijų vadovo 2 priedas),“ 2015. [Tinkle]. Prieiga internete: https://e-seimas.lrs.lt/portal/legalAct/lt/TAD/d6eb55a0948011e59c9a8f8c9980906b?jfwid=fhhu5mh5f.

[18] P. Banerjee, „Decision-Tree Classifier Tutorial,“ kaggle, 13 03 2020. [Tinkle]. Prieiga internete: https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial.

[19] A. Navlan, „Decision Tree Classification in Python Tutorial,“ datacamp, 02 2023. [Tinkle]. Prieiga internete: https://www.datacamp.com/tutorial/decision-tree-classification-python.

[20] „Decision Trees,“ scikit learn, [Tinkle]. Prieiga internete: https://scikit-learn.org/stable/modules/tree.html. [Kreiptasi kovas 2023].

[21] S. Taghvaeeyan, R. Rajamani, „Portable Roadside Sensors for Vehicle Counting, Classification, and Speed Measurement,“ *IEEE Transactions on Intelligent Transportation Systems,* t. 15, nr. 1, pp. 73-83, 2014.

[22] S. Kaewkamnerd, J. Chinrungrueng, R. Pongthornseri, S. Dumnin, „Vehicle classification based on magnetic sensor signal,“ įtraukta *The 2010 IEEE International Conference on Information and Automation,* Harbin, China, 2010.

[23] L. Breiman, „Random Forests,“ *Machine Learning,* t. 45, pp. 5-32, 2001.

[24] „Ensemble methods,“ scikit learn, [Tinkle]. Prieiga internete: https://scikit-learn.org/stable/modules/ensemble.html#random-forests. [Kreiptasi kovas 2023].

[25] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[26] V. Ghorakavi, „Neural Networks | A beginners guide,“ [Tinkle]. Prieiga internete: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/.

[27] F. La, „A Closer Look at Neural Networks,“ *MSDN Magazine,* t. 34, nr. 2, 2019.

[28] J. Brownlee, „How to Avoid Overfitting in Deep Learning Neural Networks,“ Machine Learning Mastery, 2018. [Tinkle]. Prieiga internete: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/. [Kreiptasi 2023].

[29] A. Nagpal, „L1 and L2 Regularization Methods,“ Towards Data Science, 2017. [Tinkle]. [Kreiptasi 2023].

[30] „How to choose the number of hidden layers and nodes in a feedforward neural network?,“ stackexchange, 2010. [Tinkle]. Prieiga internete: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.

Priedai

1 priedas. Naudotų požymių sąrašas

- Iš akselerometrų signalų:
  - apskaičiuotas ašių skaičius;
  - TP greitis ir apskaičiuoti atstumai tarp ašių;
  - 3 ašių koeficientas.
- Pagal magnetinių jutiklių signalus apskaičiuota TP važiavimo kryptis.
- Iš magnetinių jutiklių X ir Z ašių signalų, bei TP magnetinio parašo (pabraukti parametrai buvo skaičiuojami signalams, kurių ilgis buvo suvienodintas iki 300 verčių):
  - suma;
  - energija;
  - energija, padalinta iš apskaičiuoto TP ilgio;
  - vidurkis, standartinė deviacija, minimali ir maksimali vertės;
  - pirmos eilės išvestinės verčių ir jų kvadratų suma;
  - antros eilės išvestinės verčių ir jų kvadratų suma;
  - pirmos eilės išvestinės vidurkis, standartinė deviacija, minimali ir maksimali vertės.
- Normuota magnetinio parašo energija;
- Magnetinių jutiklių signalų X ir Y ašių teigiamų ir neigiamų verčių santykis;
- X ir Z ašių signalo bei magnetinio parašo 6 dalių vidurkiai (kaip požymiai naudotos tik 2 - 5 dalys, apskaičiuotos 2 kartus – normuotiems signalams ir ne).

2 priedas. Požymių duomenų bazės kūrimo programa

```
# %%
# %load_ext autoreload
# %autoreload 2
# %config InlineBackend.figure_format = 'png' # png, svg

# %matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import pathlib
import lib
import pandas as pd

plt.rcParams['axes.grid'] = True
plt.rcParams['axes.axisbelow'] = True
plt.rcParams['axes.xmargin'] = 0.01
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['font.size'] = 10
plt.rcParams['lines.linewidth'] = 2

distFlux = 0.071
distMag = 0.12
fs = 1000

acc_ylabel="Pagreitis, $m/s^2$"
mag_ylabel="Magnetinė indukcija, $\mu T$"

# %%
dataTable=[
['furgonai', 2],
['furgonai_p', 3],
['hecbekai', 2],
['hecbekai_e',2],
['hecbekai_p', 3],
['mazi_furgonai', 2],
['mazi_furgonai_p', 3],
['pikapai', 2],
['pikapai_p', 3],
['sedanai', 2],
['sedanai_e', 2],
['sedanai_p', 3],
['seimynines', 2],
['seimynines_p', 3],
['sunkvezimiai_2_asys', 2],
['sunkvezimiai_2_asys_mazi', 2],
['sunkvezimiai_3_asys', 3],
['sunkvezimiai_4_asys', 4],
['sunkvezimiai_5_asys', 5],
['suv', 2],
['suv_e', 2],
['suv_p', 3],
['universalai', 2],
['universalai_e', 2],
['universalai_p', 3] ]

infoDF = pd.DataFrame(columns=["type","axles"], data=dataTable)

def meanOfDividedParts(signal:np.ndarray,parts=6):
    return np.array([signal[len(signal)//parts*i:len(signal)//parts*(i+1)].mean() for i in range(parts)])

# %%
dataset = pd.DataFrame()

foldersToAnalyse = infoDF["type"].values

for folder in foldersToAnalyse:
    print(" "*80, end="\r")
    print(folder, end="\r")
    path = pathlib.PosixPath("/home/naudotvardis/Disks/HDD2/CarFiles/sorted/")/folder
    fileList = path.glob("**/*.TNK.csv")

    for i, file in enumerate(fileList):
        vehicle_type=file.parts[-3]

        flux, acc, mag, mic, cantilevers, time = lib.getData2_no_interp(file)

        speed = lib.FindSpeedAcc(acc)
        if np.isnan(speed):
            continue
```

```
        wheels = lib.FindWheels(acc)

        if(len(wheels) != infoDF.loc[infoDF["type"] == folder]["axles"].item()):
            continue

        lengths = np.diff(wheels)/1000*speed

        # region: mean of parts method from paper
        acc_demean = acc - np.mean(acc, axis=1, keepdims=True)
        accVectorLen = lib.vectorlen(acc_demean)
        accVectorLen = (accVectorLen[0]+accVectorLen[1])/2

        mag_filtered = lib.filter(mag, 15, 1000, axis=-1)
        mag_demean = mag_filtered - mag_filtered[:, -1].reshape(6, 1)  # type: ignore
        vectorLenghts = lib.vectorlen(mag_demean)
        vectorLen = (vectorLenghts[0]+vectorLenghts[1])/2
        x = (mag_demean[0]+mag_demean[3])/2
        y = (mag_demean[1]+mag_demean[4])/2
        z = (mag_demean[2]+mag_demean[5])/2

        vectorLen_c = lib.correctMagneticSignature(vectorLen, accVectorLen, method="linear")
        cut1 = np.argmax(vectorLen_c > max(vectorLen_c)/3)
        cut2 = -np.argmax(list(reversed(vectorLen_c)) > max(vectorLen_c)/5)

        vectorLen_resampled = lib.resample(vectorLen_c[cut1:cut2], 300)
        part_mean_mag = meanOfDividedParts(vectorLen_resampled)
        part_mean_mag_norm = part_mean_mag/max(abs(vectorLen_c))

        roadSide = np.sign(lib.findDelay(vectorLenghts[0], vectorLenghts[1])[0])
        if roadSide == 0:
            roadSide = 1

        x_c = lib.correctMagneticSignature(x, accVectorLen, method="linear") * roadSide
        x_resampled = lib.resample(x_c[cut1:cut2], 300)
        part_mean_x = meanOfDividedParts(x_resampled)
        part_mean_x_norm = part_mean_x/max(abs(x_c))

        z_c = lib.correctMagneticSignature(z, accVectorLen, method="linear")
        z_resampled = lib.resample(z_c[cut1:cut2], 300)
        part_mean_z = meanOfDividedParts(z_resampled)
        part_mean_z_norm = part_mean_z/max(abs(z_c))

        y_c = lib.correctMagneticSignature(y, accVectorLen, method="linear") * roadSide
        y_resampled = lib.resample(y_c[cut1:cut2], 300)
        # endregion: mean of parts method from paper

        if (x_resampled[x_resampled<0]**2).sum() != 0:
            vx = (x_resampled[x_resampled>0]**2).sum()/(x_resampled[x_resampled<0]**2).sum()
        else:
            vx = 0

        if (y_resampled[y_resampled<0]**2).sum() != 0:
            vy =  (y_resampled[y_resampled>0]**2).sum()/(y_resampled[y_resampled<0]**2).sum()
        else:
            vy = 0

        dataset = lib.pandasAppendDict(dataset, {
            "path": str(file),
            "type": vehicle_type,
            "axisnum": len(wheels),
            "length_total": lengths.sum(),
            "length_first_slot": lengths[0],
            "3axis_coeff": lengths[0]/lengths[1] if len(wheels) == 3 else -1,
            "roadSide": roadSide,
            "speed": speed,

            # region parameters from Vehicle-Classification
            "mag_resampled_sum": vectorLen_resampled.sum(),
            "mag_resampled_energy": (vectorLen_resampled**2).sum(),
            "mag_resampled_energyavg": (vectorLen_resampled**2).sum()/lengths.sum(),
            "mag_mean": vectorLen.mean(),
            "mag_std": vectorLen.std(),
            "mag_max": vectorLen.max(),
            "mag_min": vectorLen.min(),
            "mag_resampled_diff_sum": np.diff(vectorLen_resampled).sum(),
            "mag_resampled_diffE_sum": (np.diff(vectorLen_resampled)**2).sum(),
            "mag_resampled_diff2E_sum": (np.diff(vectorLen_resampled,2)**2).sum(),
            "mag_diff_max": np.diff(vectorLen).max(),
            "mag_diff_min": np.diff(vectorLen).min(),
            "mag_diff_std": np.diff(vectorLen).std(),
            "mag_diff_mean": np.diff(vectorLen).mean(),
```

```
            "x_resampled_sum": x_resampled.sum(),
            "x_resampled_energy": (x_resampled**2).sum(),
            "x_resampled_energyavg": (x_resampled**2).sum()/lengths.sum(),
            "x_mean": x.mean(),
            "x_std": x.std(),
            "x_max": x.max(),
            "x_min": x.min(),
            "x_resampled_diff_sum": np.diff(x_resampled).sum(),
            "x_resampled_diffE_sum": (np.diff(x_resampled)**2).sum(),
            "x_resampled_diff2E_sum": (np.diff(x_resampled,2)**2).sum(),
            "x_diff_max": np.diff(x).max(),
            "x_diff_min": np.diff(x).min(),
            "x_diff_std": np.diff(x).std(),
            "x_diff_mean": np.diff(x).mean(),
            "z_resampled_sum": z_resampled.sum(),
            "z_mean": z.mean(),
            "z_std": z.std(),
            "z_max": z.max(),
            "z_min": z.min(),
            "z_resampled_diff_sum": (np.diff(z_resampled)**2).sum(),
            "z_resampled_diff2_sum": (np.diff(z_resampled,2)**2).sum(),
            "z_diff_max": np.diff(z).max(),
            "z_diff_min": np.diff(z).min(),
            "z_diff_std": np.diff(z).std(),
            "z_diff_mean": np.diff(z).mean(),
            "E_normalized": np.diff(vectorLen).sum()/lengths.sum()/1000*speed,
            # endregion

            # https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6902771&tag=1
            "x_VX": vx,
            "y_VY": vy,

            # region mean of parts method from A-Field-Experience-for-a-Vehicle-Recognition-System

            "part_mean_mag_norm_2": part_mean_mag_norm[1],
            "part_mean_mag_norm_3": part_mean_mag_norm[2],
            "part_mean_mag_norm_4": part_mean_mag_norm[3],
            "part_mean_mag_norm_5": part_mean_mag_norm[4],

            "part_mean_x_norm_2": part_mean_x_norm[1],
            "part_mean_x_norm_3": part_mean_x_norm[2],
            "part_mean_x_norm_4": part_mean_x_norm[3],
            "part_mean_x_norm_5": part_mean_x_norm[4],

            "part_mean_z_norm_2": part_mean_z_norm[1],
            "part_mean_z_norm_3": part_mean_z_norm[2],
            "part_mean_z_norm_4": part_mean_z_norm[3],
            "part_mean_z_norm_5": part_mean_z_norm[4],

            "part_mean_mag_2": part_mean_mag[1],
            "part_mean_mag_3": part_mean_mag[2],
            "part_mean_mag_4": part_mean_mag[3],
            "part_mean_mag_5": part_mean_mag[4],

            "part_mean_x_2": part_mean_x[1],
            "part_mean_x_3": part_mean_x[2],
            "part_mean_x_4": part_mean_x[3],
            "part_mean_x_5": part_mean_x[4],

            "part_mean_z_2": part_mean_z[1],
            "part_mean_z_3": part_mean_z[2],
            "part_mean_z_4": part_mean_z[3],
            "part_mean_z_5": part_mean_z[4]

            # endregion
        })

dataset.to_csv("dataset_ALL.csv", index=False)
```

3 Klasifikavimo, pasinaudojant sprendimų medžiais, programa

```
import numpy as np
import matplotlib.pyplot as plt
import lib
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

plt.rcParams['axes.grid'] = True
plt.rcParams['axes.axisbelow'] = True
plt.rcParams['axes.xmargin'] = 0.01
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['font.size'] = 10
plt.rcParams['lines.linewidth'] = 2

# %%
def create_tree(x_train, y_train,**varargs):
    clf = DecisionTreeClassifier(**varargs)
    clf.fit(x_train, y_train)
    return clf

def find_best_max_features(x_train, x_test, y_train, y_test,**varargs):
    features = list(x_train.columns)
    results = []
    for i in range(1,len(features)):
        clf_entropy = create_tree(x_train, y_train, max_features=i, **varargs)
        y_pred_en = clf_entropy.predict(x_test)
        results.append(accuracy_score(y_test, y_pred_en)*100)

    return np.argmax(results)+1

# %% [markdown]
# ## Medis

# %% [markdown]
# ### Pagal ašių sk.

# %%
dataset = pd.read_csv("datasets/dataset.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset = subset.drop(index=subset[(subset['type'] == "sunkvezimiai_2_asys")|(subset['type'] ==
"seimynines")|(subset['type'] == "sunkvezimiai_2_asys_mazi")|(subset['type'] == "pikapai")|(subset['type'] ==
"furgonai")|(subset['type'] == "mazi_furgonai")].index)

subset=subset[['type','axisnum','3axis_coeff']]

subset['type'] = subset['type'].str.replace(r'su_priekaba', 'Su priekaba')
subset['type'] = subset['type'].str.replace(r'lengva', '2 ašys')

y=pd.DataFrame()
y['true'] = subset['type'].copy()
y["pred"] = np.array([lib.simple_tree(row) for index, row in subset.iterrows()])

cm = lib.getConfusionMatrix(y, square = False)
lib.plot_cm(cm, renameLabels=True, T=False, figsize=(3.5,3.5),
savefig_name="./grafikai/trecia_tarpine_ataskaita/class_paprasciausias_medis.png")

print(end="")

# %% [markdown]
# ### 2 klases

# %%
seed = 100
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)
```

```
class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
    "pikapai",
    "mazi_furgonai",
    "sunkvezimiai_2_asys_mazi",
]

class2=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1 + class2)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=102)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================================================

model = create_tree(x_train, y_train,
                        criterion='entropy', splitter='best',
                        max_features=None,
                        min_samples_leaf=30,
                        min_samples_split=60,
                        max_leaf_nodes=2,
                        max_depth=7,
                        random_state=seed,
                        class_weight='balanced')

# ============================================================================================================

true_pred = pd.DataFrame()
true_pred['true'] = y_test
true_pred['pred'] = model.predict(x_test)

cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, figsize=(2,2),
savefig_name="./grafikai/galutine/class_ilgis_matrica.png")

dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=model.feature_names_in_, class_names = model.classes_,
                impurity = False)
graph:pydotplus.graphviz.Dot = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('./grafikai/galutine/class_ilgis_medis.png')
# display(Image(graph.create_png()))

print(end="")

# %% [markdown]
# ### 3 klases ilgis

# %%
seed = 100
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

subset = subset[['type','length_total']]

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
```

```
    "seimynines",
]

class2=[
    "mazi_furgonai",
    "pikapai",
    "sunkvezimiai_2_asys_mazi",
]

class3=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2+class3)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class3)+"$)", 'Klasė III', regex=True)

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=102)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================================

model = create_tree(x_train, y_train,
                        criterion='entropy', splitter='best',
                        max_features=None,
                        min_samples_leaf=30,
                        min_samples_split=60,
                        max_leaf_nodes=4,
                        max_depth=7,
                        random_state=seed,
                        class_weight='balanced')

true_pred = pd.DataFrame()
true_pred['true'] = y_test
true_pred['pred'] = model.predict(x_test)

cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, figsize=(2.5,2.5),
savefig_name="./grafikai/galutine/class_ilgis3_matrica.png")

dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=model.feature_names_in_, class_names = model.classes_,
                impurity = False)
graph:pydotplus.graphviz.Dot = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('./grafikai/galutine/class_ilgis3_medis.png')
# display(Image(graph.create_png()))

print(end="")

# %% [markdown]
# ### 3 klases visi (permokytas)

# %%
seed = 100
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

subset = subset.drop(['length_first_slot'], axis=1)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
]
```

```
class2=[
    "mazi_furgonai",
    "pikapai",
    "sunkvezimiai_2_asys_mazi",
]

class3=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2+class3)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class3)+"$)", 'Klasė III', regex=True)

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=102)

# ============================================================================================

model = create_tree(x_train, y_train,
                        criterion='entropy', splitter='best',
                        max_features=None,
                        random_state=seed,
                        class_weight='balanced')

fig, ax = plt.subplots(1,2,figsize=(5,2.5))

true_pred = pd.DataFrame()
true_pred['true'] = y_test
true_pred['pred'] = model.predict(x_test)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[0])

true_pred = pd.DataFrame()
true_pred['true'] = y_train
true_pred['pred'] = model.predict(x_train)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")

fig.tight_layout()
fig.savefig("./grafikai/galutine/class_ilgis3_visi_permokytas_matrica.png", facecolor="white", dpi=150)

print(end="")

# %% [markdown]
# ### 3 klases visi

# %%
seed = 100
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
]

class2=[
    "mazi_furgonai",
    "pikapai",
    "sunkvezimiai_2_asys_mazi",
]

class3=[
    "furgonai",
    "sunkvezimiai_2_asys",
]
```

```
subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2+class3)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class3)+"$)", 'Klasė III', regex=True)

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=102)

# ============================================================================================

model = create_tree(x_train, y_train,
                        criterion='entropy', splitter='best',
                        max_features=None,
                        min_samples_leaf=50,
                        min_samples_split=100,
                        max_leaf_nodes=15,
                        max_depth=7,
                        random_state=seed,
                        class_weight='balanced')

fig, ax = plt.subplots(1,2,figsize=(5,2.5))

true_pred = pd.DataFrame()
true_pred['true'] = y_test
true_pred['pred'] = model.predict(x_test)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[0])

true_pred = pd.DataFrame()
true_pred['true'] = y_train
true_pred['pred'] = model.predict(x_train)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")

fig.tight_layout()
fig.savefig("./grafikai/galutine/class_ilgis3_visi_matrica.png", facecolor="white", dpi=150)

dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=model.feature_names_in_, class_names = model.classes_,
                impurity = False)
graph:pydotplus.graphviz.Dot = pydotplus.graph_from_dot_data(dot_data.getvalue())

# graph.write_png('./grafikai/galutine/class_ilgis3_visi_medis.png')
# display(Image(graph.create_png()))

print(end="")

# %%
for i,name in enumerate(model.feature_names_in_):
    if model.feature_importances_[i] > 0:
        print(f"{i:2}. {name:25s} ----> {model.feature_importances_[i]:.3f}")
print("")

# %% [markdown]
# ### Rezultatas

# %%
test_ds = np.hstack([np.array(y_test).reshape(-1,1),x_test])
test_ds = pd.DataFrame(test_ds, columns = ["type"]+list(x_test.columns))

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'_e$', '', regex=True)

listas=[
    "sunkvezimiai_3_asys",
    "sunkvezimiai_4_asys",
    "sunkvezimiai_5_asys",
    "su_priekaba"
    ]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(listas)+"$)", regex=True)]

subset = pd.concat([test_ds,subset], ignore_index=True, axis=0)
```

```
y=pd.DataFrame()
y['true'] = subset['type'].copy()
y['pred'] = np.array([lib.simple_tree(row) for _, row in subset.iterrows()])

y['pred'][y['pred'] == '2 ašys'] = model.predict(subset.iloc[:,1:][y['pred'] == '2 ašys'])

cm = lib.getConfusionMatrix(y, square = False)
lib.plot_cm(cm, renameLabels=True, T=False, figsize=(4.5,4.5),
savefig_name="./grafikai/galutine/class_paprastas_medis_VISI_matrica.png");

# %% [markdown]
# ## Miskas

# %%
dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
    "pikapai",
    "mazi_furgonai",
    "sunkvezimiai_2_asys_mazi",
]

class2=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=102)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================================

model = RandomForestClassifier(
                        criterion='entropy',
                        n_estimators=100,
                        max_features=0.2,
                        min_samples_leaf=50,
                        min_samples_split=100,
                        max_leaf_nodes=10,
                        max_depth=5,
                        random_state=103,
                        class_weight='balanced',
                        n_jobs=-1)

model.fit(x_train, y_train)

fig, ax = plt.subplots(1,2,figsize=(5,2.5))

true_pred = pd.DataFrame()
true_pred['true'] = y_test
true_pred['pred'] = model.predict(x_test)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[0])

true_pred = pd.DataFrame()
true_pred['true'] = y_train
true_pred['pred'] = model.predict(x_train)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")
```

```
fig.tight_layout()
fig.savefig("./grafikai/galutine/class_miskas_2_visi_matrica.png", facecolor="white", dpi=150)

print(end="")

# %%
dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
]

class2=[
    "mazi_furgonai",
    "pikapai",
    "sunkvezimiai_2_asys_mazi",
]

class3=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2+class3)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class3)+"$)", 'Klasė III', regex=True)

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=102)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================================

model = RandomForestClassifier(
                        criterion='entropy',
                        n_estimators=200,
                        max_features=0.5,
                        min_samples_leaf=50,
                        min_samples_split=100,
                        max_leaf_nodes=30,
                        max_depth=9,
                        random_state=103,
                        class_weight='balanced',
                        n_jobs=-1)

model.fit(x_train, y_train)

fig, ax = plt.subplots(1,2,figsize=(5,2.5))

true_pred = pd.DataFrame()
true_pred['true'] = y_test
true_pred['pred'] = model.predict(x_test)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[0])

true_pred = pd.DataFrame()
true_pred['true'] = y_train
true_pred['pred'] = model.predict(x_train)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe = ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")

fig.tight_layout()
fig.savefig("./grafikai/galutine/class_miskas_visi_matrica.png", facecolor="white", dpi=150)
```

4 Klasifikavimo, pasinaudojant neuroniniais tinklais, programa

```
import numpy as np
import matplotlib.pyplot as plt
import lib
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
import tensorflow.keras.layers as layers
from tqdm.keras import TqdmCallback

plt.rcParams['axes.grid'] = True
plt.rcParams['axes.axisbelow'] = True
plt.rcParams['axes.xmargin'] = 0.01
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['font.size'] = 10
plt.rcParams['lines.linewidth'] = 2

# %% [markdown]
# # 2 tipu TP

# %% [markdown]
# ## normalus

# %%
seed = 100
tf.keras.utils.set_random_seed(seed)
tf.random.set_seed(seed)
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
    "pikapai",
    "mazi_furgonai",
    "sunkvezimiai_2_asys_mazi",
]

class2=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)

modelLabelEncoder = LabelEncoder()
subset['type'] = modelLabelEncoder.fit_transform(subset['type'])

scaler = StandardScaler()
subset[subset.columns[1:]] = scaler.fit_transform(subset.iloc[:, 1:])

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=100)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=100)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================

model = tf.keras.Sequential([
    layers.Dense(len(subset.columns), activation='relu', input_shape=(len(subset.columns)-1,),
kernel_regularizer=tf.keras.regularizers.L1(0.001)),
    layers.Dropout(.4),
    layers.Dense(len(subset['type'].unique()), activation='softmax')
```

```
])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, class_weight=weigths, epochs=30, batch_size=32*6, validation_data=(x_val,
y_val), verbose=0, callbacks=[TqdmCallback(verbose=1)])

# %%
fig,ax = plt.subplots(1,2,figsize=(10,3.5),dpi = 100)

ax[0].plot(history.history['accuracy'], label='accuracy')
ax[0].plot(history.history['val_accuracy'], label='accuracy')
ax[1].plot(history.history['loss'], label='loss')
ax[1].plot(history.history['val_loss'], label='val_loss')

fig, ax = plt.subplots(1,2,figsize=(5,2.5))

test_dataset = pd.DataFrame(x_test)
test_dataset['type'] = y_test.to_numpy()
train_dataset = pd.DataFrame(x_train)
train_dataset['type'] = y_train.to_numpy()

true_pred = lib.calcTrueAndPred_tf(test_dataset, model, modelLabelEncoder)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe=ax[0])

true_pred = lib.calcTrueAndPred_tf(train_dataset, model, modelLabelEncoder)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe=ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")

fig.align_labels()
fig.tight_layout()

fig.savefig("grafikai/galutine/NN_TF_2_matrica.png", dpi=150)

# %% [markdown]
# ## permokytas

# %%
seed = 100
tf.keras.utils.set_random_seed(seed)
tf.random.set_seed(seed)
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
    "pikapai",
    "mazi_furgonai",
    "sunkvezimiai_2_asys_mazi",
]

class2=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)

modelLabelEncoder = LabelEncoder()
subset['type'] = modelLabelEncoder.fit_transform(subset['type'])

scaler = StandardScaler()
subset[subset.columns[1:]] = scaler.fit_transform(subset.iloc[:, 1:])

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=100)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=100)
```

```
values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ==============================================================================

model_overfit = tf.keras.Sequential([
    layers.Dense(len(subset.columns), activation='relu', input_shape=(len(subset.columns)-1,)),
    layers.Dense(len(subset['type'].unique()), activation='softmax')
])
model_overfit.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history_overfit = model_overfit.fit(x_train, y_train, class_weight=weigths, epochs=50, batch_size=32*6,
validation_data=(x_val, y_val), verbose=0, callbacks=[TqdmCallback(verbose=1)])

# %%
fig,ax = plt.subplots(2,figsize=(6,4))

ax[0].plot(history_overfit.history['loss'], label='Treniravimo')
ax[0].plot(history_overfit.history['val_loss'], label='Validavimo')
ax[0].set(title="Permokyto modelio praradimo funkcija", xlabel="Epchos", ylabel="Praradimas")
ax[0].legend()

ax[1].plot(history.history['loss'], label='Treniravimo')
ax[1].plot(history.history['val_loss'], label='Validavimo')
ax[1].set(title="Nepermokyto modelio praradimo funkcija", xlabel="Epchos", ylabel="Praradimas")
ax[1].legend()

fig.align_labels()
fig.tight_layout()

# %% [markdown]
# # 3 tipu TP

# %% [markdown]
# ### bandymas itraukiant visas klases

# %%
seed = 102
tf.keras.utils.set_random_seed(seed)
tf.random.set_seed(seed)
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
    "maziausios_lengviausios",
]

class2=[
    "mazi_furgonai",
    "pikapai",
    "sunkvezimiai_2_asys_mazi",
]

class3=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2+class3)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class3)+"$)", 'Klasė III', regex=True)

modelLabelEncoder = LabelEncoder()
subset['type'] = modelLabelEncoder.fit_transform(subset['type'])

scaler = StandardScaler()
subset[subset.columns[1:]] = scaler.fit_transform(subset.iloc[:, 1:])
```

```
x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=100)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=100)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================

model = tf.keras.Sequential([
    layers.Dense(len(subset.columns), activation='relu', input_shape=(len(subset.columns)-1,),
kernel_regularizer=tf.keras.regularizers.L1(0.001)),
    layers.Dropout(.4),
    layers.Dense(50, activation='relu', kernel_regularizer=tf.keras.regularizers.L2(0.2)),
    layers.Dense(len(subset['type'].unique()), activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, class_weight=weigths, epochs=150, batch_size=32*6, validation_data=(x_val,
y_val), verbose=0, callbacks=[TqdmCallback(verbose=1)])

# %%
fig,ax = plt.subplots(1,2,figsize=(10,3.5),dpi = 100)

ax[0].plot(history.history['accuracy'], label='accuracy')
ax[0].plot(history.history['val_accuracy'], label='accuracy')
ax[1].plot(history.history['loss'], label='loss')
ax[1].plot(history.history['val_loss'], label='val_loss')

fig,ax = plt.subplots(1,2,figsize=(5,2.5), dpi = 150)

test_dataset = pd.DataFrame(x_test.copy())
test_dataset['type'] = y_test.to_numpy()
train_dataset = pd.DataFrame(x_train.copy())
train_dataset['type'] = y_train.to_numpy()

true_pred = lib.calcTrueAndPred_tf(test_dataset, model, modelLabelEncoder)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe=ax[0])

true_pred = lib.calcTrueAndPred_tf(train_dataset, model, modelLabelEncoder)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe=ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")

fig.align_labels()
fig.tight_layout()
fig.savefig("./grafikai/galutine/NN_TF_3klases_matrica.png", dpi=150)

tf.keras.utils.plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=False, rankdir='LR')

# %%
test_ds = np.hstack([np.array(modelLabelEncoder.inverse_transform(y_test)).reshape(-1,1),
scaler.inverse_transform(x_test)])
test_ds = pd.DataFrame(test_ds, columns = ["type"]+list(x_test.columns))

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'_e$', '', regex=True)

listas=[
    "sunkvezimiai_3_asys",
    "sunkvezimiai_4_asys",
    "sunkvezimiai_5_asys",
    "su_priekaba"
    ]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(listas)+"$)", regex=True)]
subset = pd.concat([test_ds,subset], ignore_index=True, axis=0)

y=pd.DataFrame()
y['true'] = subset['type'].copy()
y['pred'] = np.array([lib.simple_tree(row) for _, row in subset.iterrows()])

asys2 = subset.iloc[:,1:][y['pred'] == '2 ašys']
asys2 = scaler.transform(asys2)
```

```
y['pred'][y['pred'] == '2 ašys'] = modelLabelEncoder.inverse_transform(np.argmax(model.predict(asys2, verbose=0),
axis=1))

cm = lib.getConfusionMatrix(y, square = False)
lib.plot_cm(cm, renameLabels=True, T=False, figsize=(4.5,4.5),
savefig_name="./grafikai/galutine/NN_VISI_matrica.png");

# %% [markdown]
# ##### patikrinimas kas kiek teisingai suklasifikuoja

# %%
dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset = dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)

listas = [
    "sedanai",
    "hecbekai",
    "furgonai",
    "pikapai",

    "sunkvezimiai_2_asys_mazi",
    "suv",
    "universalai",
    "sedanai_e",
    "mazi_furgonai",
    "hecbekai_e",
    "suv_e",
    "seimynines",
    "maziausios_lengviausios",
    "maziausios_lengviausios_e",
    "sunkvezimiai_2_asys",
    "universalai_e"
    ]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(listas)+"$)", regex=True)]
le = LabelEncoder()
subset['type'] = le.fit_transform(subset['type'])
subset[subset.columns[1:]] = scaler.transform(subset[subset.columns[1:]])

true_pred = lib.calcTrueAndPred_tf(subset, model, (le, modelLabelEncoder))
cm = lib.getConfusionMatrix(true_pred, square = False)
lib.plot_cm(cm, renameLabels=True, T=False, figsize=(8,8))

# %% [markdown]
# ## Kintamuju svarbumas

# %%
seed = 102
tf.keras.utils.set_random_seed(seed)
tf.random.set_seed(seed)
np.random.seed(seed)


dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
subset=dataset.copy()

subset = subset[['type', 'axisnum', 'length_total', '3axis_coeff',
                    'mag_diff_std', 'x_resampled_energyavg', 'x_mean', 'x_std',
       'x_max', 'x_min', 'x_resampled_diff_sum', 'x_resampled_diffE_sum',
       'x_resampled_diff2E_sum', 'x_diff_max', 'x_diff_min', 'x_diff_std',
       'x_diff_mean', 'z_resampled_sum', 'z_mean', 'z_std', 'z_max', 'z_min',
       'z_resampled_diff_sum', 'z_resampled_diff2_sum', 'z_diff_max',
       'z_diff_min', 'z_diff_std', 'z_diff_mean', 'E_normalized', 'x_VX',
       'y_VY',
                    'part_mean_mag_norm_5', 'part_mean_z_norm_5',
'part_mean_mag_3','part_mean_x_2','part_mean_z_3']]

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'._e$', '', regex=True)

class1=[
    "sedanai",
    "hecbekai",
    "suv",
    "universalai",
    "seimynines",
    "maziausios_lengviausios",
]

class2=[
```

```
    "mazi_furgonai",
    "pikapai",
    "sunkvezimiai_2_asys_mazi",
]

class3=[
    "furgonai",
    "sunkvezimiai_2_asys",
]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(class1+class2+class3)+"$)", regex=True)]

subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class1)+"$)", 'Klasė I', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class2)+"$)", 'Klasė II', regex=True)
subset['type'] = subset['type'].str.replace('(?:^'+'$)|(?:^'.join(class3)+"$)", 'Klasė III', regex=True)

modelLabelEncoder = LabelEncoder()
subset['type'] = modelLabelEncoder.fit_transform(subset['type'])

scaler = StandardScaler()
subset[subset.columns[1:]] = scaler.fit_transform(subset.iloc[:, 1:])

x_train, x_test, y_train, y_test = train_test_split(subset.iloc[:, 1:], subset.iloc[:, 0], test_size=0.3,
random_state=100)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=100)

values_counts = y_train.value_counts().sort_index()
weigths = 1/(values_counts.values/(max(values_counts.values)))
weigths =  dict(zip(values_counts.index, weigths))

# ============================================================================

model = tf.keras.Sequential([
    layers.Dense(len(subset.columns), activation='relu', input_shape=(len(subset.columns)-1,),
kernel_regularizer=tf.keras.regularizers.L1(0.001)),
    layers.Dropout(.4),
    layers.Dense(200, activation='relu', kernel_regularizer=tf.keras.regularizers.L2(0.2)),
    layers.Dense(len(subset['type'].unique()), activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, class_weight=weigths, epochs=50, batch_size=32*6, validation_data=(x_val,
y_val), verbose=0, callbacks=[TqdmCallback(verbose=1)])

fig,ax = plt.subplots(1,2,figsize=(10,3.5),dpi = 100)

ax[0].plot(history.history['accuracy'], label='accuracy')
ax[0].plot(history.history['val_accuracy'], label='accuracy')
ax[1].plot(history.history['loss'], label='loss')
ax[1].plot(history.history['val_loss'], label='val_loss')

fig,ax = plt.subplots(1,2,figsize=(5,2.5), dpi = 150)

test_dataset = pd.DataFrame(x_test.copy())
test_dataset['type'] = y_test.to_numpy()
train_dataset = pd.DataFrame(x_train.copy())
train_dataset['type'] = y_train.to_numpy()

true_pred = lib.calcTrueAndPred_tf(test_dataset, model, modelLabelEncoder)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe=ax[0])

true_pred = lib.calcTrueAndPred_tf(train_dataset, model, modelLabelEncoder)
cm = lib.getConfusionMatrix(true_pred, square = True)
lib.plot_cm(cm, renameLabels=False, T=False, axe=ax[1])

ax[0].set_title("Testavimo duomenys")
ax[1].set_title("Apmokymo duomenys")

fig.align_labels()
fig.tight_layout()

# %% [markdown]
# # Apibendrinimas

# %%
seed = 99
tf.keras.utils.set_random_seed(seed)
tf.random.set_seed(seed)
np.random.seed(seed)

dataset = pd.read_csv("datasets/dataset_ALL.csv").iloc[:, 1:]
```

```
subset=dataset.copy()

subset['type'] = subset['type'].str.replace(r'.*_p', 'su_priekaba', regex=True)
subset['type'] = subset['type'].str.replace(r'_e$', '', regex=True)

listas=[
    "sedanai",
    "hecbekai",
    "universalai",
    "suv",
    "seimynines",
    "pikapai",
    "mazi_furgonai",
    "sunkvezimiai_2_asys_mazi",
    "furgonai",
    "sunkvezimiai_2_asys",
    "sunkvezimiai_3_asys",
    "sunkvezimiai_4_asys",
    "sunkvezimiai_5_asys",
    "su_priekaba"
    ]

subset = subset[subset["type"].str.contains('(?:^'+'$)|(?:^'.join(listas)+"$)", regex=True)]

lib.rename(subset)

y=pd.DataFrame()
y['true'] = subset['type'].copy()
y['pred'] = np.array([lib.simple_tree(row) for _, row in subset.iterrows()])

transformed = scaler.transform(subset.iloc[:,1:][y['pred'] == 'Lengva'])
NN_Mod = modelLabelEncoder.inverse_transform(np.argmax(model.predict(transformed, verbose=0), axis=1))
y['pred'][y['pred'] == 'Lengva'] = NN_Mod

cm = lib.getConfusionMatrix(y, square = False)
lib.plot_cm(cm, renameLabels=False, T=False, figsize=(7,7))
```

5 Taikytų metodų funkcijos

```
import numpy as np
import pandas as pd
from scipy.interpolate import interp1d
from scipy.signal import find_peaks

def correctMagneticSignature(mag: np.ndarray, accVectorLen: np.ndarray, method="cubic", th=1.25, samples="all",
lin_filter_f=30, lin_filter_fs=1000, acc_fs = 1000) -> np.ndarray:
    from scipy.interpolate import interp1d
    wheelZones = signalHelper.filter(accVectorLen, 10, fs=acc_fs)

    croppedSignature = mag[wheelZones < wheelZones.mean()*th]

    if samples == "all":
        time = np.linspace(0, 1, len(wheelZones), endpoint=False)
    else:
        time = np.linspace(0, 1, samples, endpoint=False)  # type: ignore

    timePart = np.linspace(0, 1, len(wheelZones), endpoint=False)[wheelZones < wheelZones.mean()*th]

    # in case wheels are at the beginning of recording
    offset = 10
    croppedSignature[:offset] = mag[:offset]
    timePart[:offset] = time[:offset]

    croppedSignature[-offset:] = mag[-offset:]
    timePart[-offset:] = time[-offset:]

    # print(timePart.shape, croppedSignature.shape)
    result = interp1d(timePart, croppedSignature, kind=method)(time)

    if method == "linear":
        result = signalHelper.filter(result, lin_filter_f, fs=lin_filter_fs)

    return result

def meanOfDividedParts(signal: np.ndarray, parts=6):
    return np.array([signal[len(signal)//parts*i:len(signal)//parts*(i+1)].mean() for i in range(parts)])

def simple_tree(df_row):
    if (df_row["axisnum"]==2):
        return "2 ašys"
    elif(df_row["axisnum"]==4):
        return "sunkvezimiai_4_asys"
    elif(df_row["axisnum"]==5):
        return "sunkvezimiai_5_asys"
    elif (df_row["axisnum"]==3):
        return "su_priekaba" if (df_row["3axis_coeff"]<=1.5) else "sunkvezimiai_3_asys"
    else:
        "None"


def FindWheels(acc, filterF=7, fs=1000, logar=True):
    accData = acc.copy()
    accData -= np.mean(accData, axis=1, keepdims=True)
    acc_vect_len = vectorlen(accData)
    acc_vect_len = (acc_vect_len[0] + acc_vect_len[1])/2

    logar = np.log(acc_vect_len.copy())/np.log(5) if logar else acc_vect_len.copy()
    filt = filter(logar.copy(), filterF, fs=fs)
    filt_mean = filt.mean()
    return sig.find_peaks(filt, filt_mean+abs(filt_mean)*0.3, distance=len(filt)/20)[0]

def FindSpeedAcc(acc: np.ndarray, distanceAccGrooves=0.43, fs=1000):
    accData = acc.copy()
    accData -= np.mean(accData, axis=1, keepdims=True)
    acc_vect_len = vectorlen(accData)
    acc_vect_len = (acc_vect_len[0] + acc_vect_len[1])/2

    filteredAcc = filter(acc_vect_len, 30, fs=fs)
    threshold = filteredAcc.mean()*1.1
    peaks = sig.find_peaks(filteredAcc, threshold, prominence=0)[0]

    peakTimes = peaks/fs

    continuity_check_array = np.where(filteredAcc > threshold, 0, np.nan)

    speed_acc = np.array([])
    i = 0
    while i < len(peaks)-1:
        if(not np.isnan(continuity_check_array[peaks[i]:peaks[i+1]]).any()):
```

```
            j = 1
            while((i+j < len(peaks)-1) and not np.isnan(continuity_check_array[peaks[i+j]:peaks[i+j+1]]).any()):
                j += 1
            i += j
            if(j == 1):
                speed_acc = np.append(speed_acc, distanceAccGrooves / (peakTimes[i] - peakTimes[i-1]))
        i += 1

    return speed_acc.mean() if(speed_acc.size > 0) else np.nan

def resample(y: np.ndarray, samples: float, method="cubic") -> np.ndarray:
    from scipy.interpolate import interp1d

    return interp1d(np.linspace(0, 1, len(y), endpoint=False), y, kind=method,
fill_value="extrapolate")(np.linspace(0, 1, int(samples), endpoint=False))

def resampleX(y: list[np.ndarray], samples: float, method="cubic") -> np.ndarray:
    return np.array([resample(array, samples, method) for array in y])
```