VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
DUOMENŲ MOKSLO BAKALAURO STUDIJŲ PROGRAMA

Duomenų mokslo projektas - kursinis darbas

Automatinis aritmijų aptikimas naudojant elektrokardiogramos signalus

(Automatic arrhythmia detection from electrocardiogram signals)

Atliko: 3 kurso 2 grupės studentai
Vainius Gataveckas (parašas)
Dovydas Martinkus (parašas)

Darbo vadovas:
doc. dr. Povilas Treigys (parašas)

Vilnius
2022

Turinys

Automatinis aritmijų aptikimas naudojant elektrokardiogramos signalus

Santrauka

Šiame darbe atliekamas automatinis aritmijų aptikimas naudojant kintamo ilgio individualių širdies pūpsnių elektrokardiogramos duomenis. Širdies pūpsniams atliktas pradinis apdorojimas naudojant FIR ir medianos filtrus. Požymiai išskirti naudojantis bangelių transformacija, Ermito polinomų koeficientais, aukštesnės eilės statistikomis, morfologinėmis/statistinėmis pūpsnių savybėmis. Klasifikavimui atlikti naudoti ir tarpusavyje palyginti logistinės regresijos ir atraminių vektorių mašinos metodai (antrajam metodui naudojant dvi skirtingas galutinio sprendimo schemas). Kombinuojant išskirtų požymių rinkinius abiems klasifikavimo metodams pagerinti atitinkamų lyginamųjų modelių, vietoje išskirtų požymių naudojančių patį suvienodinto ilgio elektrokardiogramos signalą, rezultatai. Atraminių vektorių mašinos metodu gautas geresnis klasifikavimo tikslumas, bet patologinių klasių aptikimo atžvilgiu konservatyvūs rezultatai lyginant su logistinės regresijos metodu. Tyrime taip pat eksperimentiškai įvertinta pradinio apdorojimo įtaka klasifikavimo tikslumui, atskirai įvertinta surastų pūpsnių išskirčių klasifikavimo kokybė.

Raktiniai žodžiai: Artimijos; Elektrokardiograma; Širdies pūpsnių klasifikavimas; Požymių išskyrimas; Logistinė regresija; Atraminių vektorių mašina

Automatic arrhythmia detection from electrocardiogram signals

Abstract

This work presents automatic arrhythmia detection from varying-length electrocardiogram signals of individual heartbeats. The signals were preprocessed by applying FIR and median filters. Features extraction from the signal was done by the use of wavelet transformation, Hermite function coefficients, higher order statistics as well as statistical/morphological properties of the heartbeat signal. Logistic regression and Support Vector Machine methods (utilizing two different final decision schemes for the latter) were used to classify the heartbeats. For both methods the usage of combinations of extracted feature groups improved on the results achieved by the respective baseline models based on fixed length raw signal input. The approach based on Support Vector Machine showed significantly better perfomance, but conservative results in terms of classifying heartbeats as pathological compared to logistic regression. The effect of applied preprocessing steps on the perfomance of the classifiers and the classification of identified outlier heatbearts were experimentally evaluated.

Key words: Arrhythmias; Electrocardiogram; Heartbeat classification; Feature extraction; Logistic regression; Support Vector Machine

Įvadas

Pasak PSO (Pasaulio Sveikatos Organizacijos) kardiovaskulinės ligos yra dažniausiai pasitaikanti mirčių priežastis pasaulyje. Dėl šių lygų gyvybę praranda apytiksliai 17.9 milijonų žmonių kiekvienais metais [RMJ+20]. Ši problema tampa vis ryškesnė: fiksuotų kardiovaskulinių ligų kiekis per 29 metus (1990-2019 metais) padidėjo beveik dvigubai nuo 271 iki 523 milijonų [RMJ+20]. Dėl anksčiau minėtų priežasčių aktualus kardiovaskulinių ligų aptikimas ir šiam tikslui gali būti pasitelkiami statistiniai ar mašininio mokymo modeliai. Prieš tai minėti modeliai gali būti sudaromi naudojant elektrokardiogramos (širdies elektrinės veiklos įrašo) duomenimis.

Šio darbo tikslas yra naudojant individualių kintamo ilgio širdies pūpsnių elektrokardiogramos duomenis išskirti požymius širdies pūpsniams ir naudojant šiuos požymius klasifikuoti širdies pūpsnius kaip sveikus arba priklausančius tam tikrai aritmijų (širdies ritmo sutrikimų) klasei.

Šiam tikslui pasiekti iškelti uždaviniai:

- Literatūros apžvalgos atlikimas, srityje naudojamų metodų privalumų ir trūkumų įvertinimas.
- Pirminės duomenų analizės atlikimas.
- Širdies pūpsnių požymių išskyrimo metodų parinkimas.
- Aritmijų klasifikavimo metodų parinkimas.
- Modelių, naudojančių aprašytus požymių išskyrimo ir klasifikavimo metodus, sudarymas.
- Modelių eksperimentinis įvertinimas.

Įprastinė automatinė aritmijų aptikimo iš elektrokardiogramos signalo sistema sudaryta širdies pūpsnių segmentavimo, signalo pradinio apdorojimo, požymių išskyrimo iš kiekvieno širdies pūpsnio ir pateikimo klasifikatoriui, kuris nustato širdies pūpsnio tipą [COR04; dSSC+16]. Šis darbas daugiausiai orientuotas į požymių išskyrimo ir klasifikavimo dalis.

Tyrimą sudaro teorinė ir eksperimentinė dalys. Teorinėje dalyje pirmiausia aprašoma temos specifika, paaiškinami naudojami terminai. Literatūros apžvalgoje pristatomi su automatiniu aritmijų aptikimu susiję tyrimai. Galiausiai aprašomi naudojami duomenys, atliekamas teorinis supažindinimas su naudotais metodais. Eksperimentinė dalis pradedama pirmine duomenų analize. Vėliau aprašomos naudotų požymių išskyrimo ir klasifikavimo metodų specifikos, aprašomi sudaryti modeliai ir jų eksperimentiniai įvertinimai. Pabaigoje pateikiamos išvados ir rekomendacijos.

1. Probleminės srities analizė

1.1. Širdies aritmijos

Širdies aritmijos yra bet kokie normalaus širdies ritmo sutrikimai, pavyzdžiui, per lėtas, per greitas ar ireguliarus širdies ritmas [EX21]. Dalis galimų aritmijų simptomų yra palpitacija, jaučiama pauzė tarp širdies pūpsnių, sunkesniais atvejais galimas alpimas, krūtinės skausmai. Aritmijų tipai gali pasireikšti ir be simptomų. Laikinos ar retai pasireiškiančios aritmijos dažniausiai nėra pavojingos sveikatai, tačiau dėl kai kurių aritmijų gali vystytis infarktas, insultas, išsivystyti širdies nepakankamumas ar kitos širdies ligos. Retais atvejais aritmijos gali tapti staigios mirties priežastimi [COR04]. Širdies aritmijos kyla dėl elektrinio impulso plitimo širdyje sutrikimų [GP12]. Aritmijų gydymui galimas gydymas medikamentais, širdies stimuliatoriaus naudojimas, širdies operavimas. Pagal širdies susitraukimo dažnį aritmijos gali būti skirstomos į padidinto, normalaus ar reto dažnio. Kitas skirstymo metodas pagal aritmijų kilimo širdyje vietą jas skirsto į prieširdinies (angl. supraventricular) ir skilvelines (angl. ventricular) aritmijas [HPA18]. Šis (antrasis) aritmijų skirstymo būdas pasitelkiamas tyrime.

1.2. Elektrokardiograma (EKG)

Elektrokardiograma yra širdies elektrinės veiklos per tam tikrą laiką įrašas. Šios informacijos analizė gali pateikti svarbios informacijos apie paciento širdies sveikatos būklę, įskaitant ir aritmijų aptikimą [GP12]. Įprasta elektrokardiograma atliekama ant paciento odos specialiose vietose padedant tam tikrą elektrodų kiekį [EX21].

1961 metais Norman J. Holter sukūrė pirmą sistemą nenustojamam elektrokardiogramos įrašymui ambulatorinėmis sąlygomis. Ilgesnės trukmės elektrokardiograma, reikalinga klinikinėms širdies ligų diagnozėms, dažniausiai gaunama naudojant šį (Holterio) monitoravimą, trunkantį 24, 48 arba 72 valandas [MNR$^+$19].

Kiekvieną širdies pūpsnį elektrokardiogramoje sudaro nuokrypiai nuo pradinės linijos vadinami bangomis, kurie parodo širdies elektrinės veiklos kitimą [MNR$^+$19]. Aritmijų tipai gali būti aptinkami analizuojant skirtumus šiose bangose [dSSC$^+$16; MNR$^+$19]. Iš elektrokardiogramą sudarančių bangų trys svarbiausios yra [EX21]:

- P banga, kuri atitinka prieširdžių depolarizaciją.
- QRS kompleksas, kuris yra sudarytas iš Q, R ir S bangų. QRS kompleksas atitinka skilvelių depolarizaciją ir pasiekia didžiausią amplitudę R taške.
- T banga, kuri atitinka skilvelių repolarizaciją.

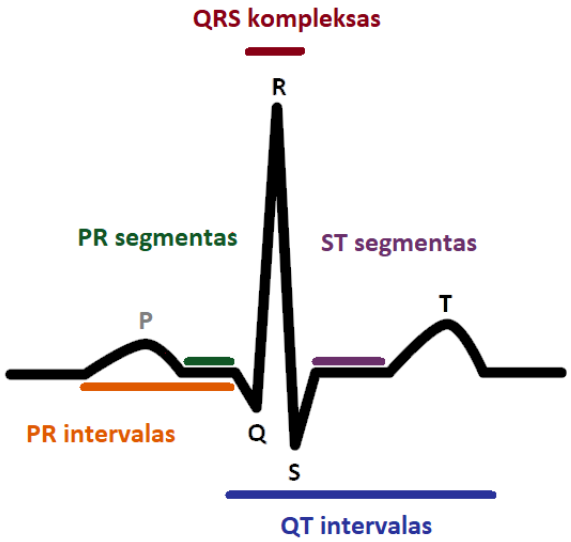
Šių bangų pradžios, pabaigos ir aukščiausių vietų taškai kartu yra vadinamos elektrokardiogramos charakteristiniais taškais (angl. characteristic points).

Kai kuriais atvejais (maždaug ketvirtadalyje populiacijos) po T bangos gali kilti papildoma U banga [GP12; MNR$^+$19].

Kiti svarbūs širdies pūpsnio komponentai elektrokardiogramoje yra [EX21]:

- PR segmentas matuojamas nuo P bangos pabaigos iki QRS komplekso pradžios ir atitinka elektrinio impulso judėjimą į skilvelius.
- PR intervalas matuojamas nuo P bangos pradžios iki QRS komplekso pradžios.
- ST segmentas matuojamas nuo QRS komplekso pabaigos iki T bangos pradžios ir atitinka visišką skilvelių depolarizaciją.
- QT intervalas matuojamas nuo QRS komplekso pradžios iki T bangos pabaigos.
- RR intervalas matuojamas tarp dviejų R taškų ir atitinka vieną pilną širdies ciklą.

Visi šie elektrokardiogramos komponentai pavaizduoti grafiškai (1 pav.).



1 pav. Normalaus širdies pūpsnio komponentai elektrokardiogramoje [IBA19]

1.3. Automatinio aritmijų aptikimo sistemos

Norint aptikti širdies ligas, elektrokardiograma turi būti ištirta gydytojo kardiologo. Tačiau turint didelį duomenų kiekį tai daug laiko užtrunkantis procesas ir nustatyta diagnozė gali būti pagrįsta tik maža dalimi išanalizuotų širdies pūpsnių [EX21; LFD+11]. Dėl šios priežasties per paskutinius dešimtmečius daug tyrėjų dėmesio sulaukė automatizuotos patologinių širdies pūpsnių klasifikavimo sistemos [MNR+19].

Praktinis aritmijų aptikimo automatizavimas yra sudėtingas dėl kelių priežasčių [LFD+11]:

- Elektrokardiogramos signalas realiose situacijose yra paveiktas triukšmo, susidarančio dėl blogo elektrodų kontakto, kvėpavimo ar kitų priežasčių.
- Tik maža dalis širdies pūpsnių yra patologiniai, nors jų automatinis aptikimas yra svarbiausias.
- Aritmijų klasifikavimo metodams reikalingas požymių išskyrimas iš elektrokardiogramos signalo.

1.4. Literatūros apžvalga

Lannoy et al. [LFD+11] atliktame tyrime autoriai išskyrė požymius naudodamiesi iš anksto anotuotais charakteristiniais širdies pūpsnio taškais (kiekvienai iš QRS, T ir P bangų apskaičiuotos didžiausia ir mažiausios reikšmės, plotas, standartinis nuokrypis, asimetrijos koeficientas, ekscesas, taip pat panaudoti širdies pūpsnį sudarančių intervalų ilgiai), RR intervalais (laikas iki praėjusio pūpsnio, laikas iki sekančio pūpsnio, vidutinis laikas tarp pūpsnių 10 pūpsnių lange).

Morfologinės (su signalo forma susijusios) savybės gautos iš pūpsnį sudarančių bangų paimant keleto joms priklausančių taškų amplitudės reikšmes. Kitos požymių grupės išskirtos naudojantis Ermito bazės funkcijų koeficientais, taip pat tam tikrų laipsnių kumulantais. Papildomai išskirtos ir normalizuotos požymių versijos charakteristiniais taškais ir RR intervalais pagrįstiems požymiams, normalizuojant naudojantis vidutinėmis šių požymių reikšmėmis tiriamam pacientui.

Akivaizdu, kad norint išskirti normalizuotus požymius būtina turėti ankstesnius duomenis apie tiriamą pacientą. Norint išskirti RR intervalų vidurkį lange aplink tiriamą pūpsnį sąlyga dar griežtesnė: būtina turėti duomenis apie iš eilės einančius tiriamo paciento širdies pūpsnius.

Klasifikavimas atliktas naudojant atraminių vektorių mašiną (angl. Support Vector Machine, toliau - SVM). SVM pritaikytas daugelio klasių uždaviniui (angl. multiclass) naudojant vienas-prieš-vieną (angl. One-Versus-One, toliau - OVO) schemą. Išbalansuotų klasių problemai spręsti naudotas svorinis SVM, kiekvienai klasei keičiant tikslo funkciją. Požymių svarbumas vertintas aplanko metodu, perrenkant visas požymių grupių kombinacijas, su jomis sudarant SVM modelį ir įvertinant klasifikavimo kokybę.

Mondéjar-Guerra et al. [MNR+19] tyrime taip pat naudoti RR intervalais pagrįsti požymiai (tiek paprasti, tiek normalizuoti). Autorių aprašytas morfologinis kriterijus vietoje grynų amplitudžių reikšmių yra pagrįstas Euklidiniais atstumais tarp R taško ir tam tikrais atstumais nuo šio taško parinktų maksimumo taškų.

Kitos požymių grupės išskirtos naudojantis bangelių transformacijos koeficientais (angl. wavelet transform), taip pat dalijant pūpsnį į 5 dalis ir apskaičiuojant eksceso (angl. kurtosis) ir asimetrijos koeficiento (angl. skeweness) reikšmes kiekvienam intervalui.

Priešingai negu prieš tai aprašytame straipsnyje, aprašytam požymių rinkiniui išskirti nėra reikalinga aptikti QRS komplekso, P ir T bangų vietos ar trukmės (kas gali būti sudėtinga jeigu duomenyse iš anksto nėra šių taškų vietos anotacijų), pakanka tik R taško aptikimo.

Aritmijų klasifikavimui taip pat pasirinkta naudoti SVM su OVO schema. Išbalansuotų klasių problemai kiekvienam spręsti kiekvienam SVM modeliui naudoti svoriai lygūs dviejų klasių stebėjimų skaičiaus santykiui. Kaip ir prieš tai aprašytame tyrime, požymiai vertinti naudojant perrinkimą. Autorių atliktas visiems požymiams apmokyto SVM ir vienam požymiui apmokytų SVM ansamblių (angl. ensemble) eksperimentinis palyginimas parodė antrojo metodo pranašumą.

Kitaip negu ankščiau paminėtuose tyrimuose, Mar et al. [MML+11] atliktame tyrime, statistiniai (tokie kaip dispersija, asimetrijos koeficientas, ekscesas, histogramos dispersija) ir morfologiniai (tokie kaip mažiausios ir didžiausios reikšmės, jų santykis, pūpsnio plotas) požymiai išskirti ne tik iš širdies pūpsnį sudarančių bangų, bet ir iš viso pūpsnio. Tam tikri morfologiniai ir statistiniai požymiai atskirai apskaičiuoti skirtingiems signalo lygmenims, gautiems naudojant bangelių transformaciją.

Aritmijų klasifikavimas atliktas naudojant tiesinio diskriminanto (angl. linear discriminant) klasifikatorių ir dirbtinius neuroninius tinklus (angl. artificial neural networks) pasitelkiant daugiasluoksnio perceptrono (angl. multilevel perceptron, toliau - MLP) architektūrą. Geresni rezultatai gauti naudojant MLP klasifikatorių.

Požymiai išskirti naudojant bangelių transformaciją ir Guler at al. [İE05]. Bangelių transformacija gautų koeficientų dimensija sumažinta kiekvieno lygmens koeficientams apskaičiavus tokius apibendrinančius dydžius kaip jų absoliučių reikšmių vidurkį, standartinį nuokrypį, absoliučių reikšmių vidurkių santykį gretimiems lygmenims.

Aritmijų aptikimui pasitelkiami ir dimensijos mažinimu pagrįsti požymių išskyrimo metodai: požymiams išskirti naudojama pagrindinių komponenčių analizė (angl. Principal Component Analysis) [KSS+09], nepriklausomų komponenčių analizė (angl. Independent Component Analysis) [YCK10], atsitiktinės projekcijos (angl. random projections) [HLZ+14]. Akivaizdu, kad norint panaudoti šiuos metodus turint kintamo ilgio širdies pūpsnius susiduriama su problemomis, nes neturimas pastovus skaičius požymių, kurių dimensija galėtų būti mažinama.

Egzistuoja ir kiti, rečiau nagrinėtoje literatūroje naudojami požymių išskyrimo metodai, pavyzdžiui, diskrečios kosinusinės transformacijos koeficientai (angl. discrete cosine transform) [MAL+13], požymiai, gaunami iš Integrate and Fire biologinio neurono modeliu sudarytos laiko eilutės [ALP12].

Vėliausiai atliktuose tyrimuose daugiau dėmesio sulaukė dirbtiniais neuroniniais tinklais pagrįstas požymių išskyrimas: Sallami et al. [SH18] elektrokardiogramos signalui panaudota konvoliucinių neuroninių tinklų (angl. convolutional neural networks, toliau - CNN) architektūra. Autoriai pasirinko rankiniu būdu neišskirti jokių požymių. Essa et al. [EX21] naudotas hibridinis metodas, naudojantis tiek rankiniu būdu išskirtus (RR intervalais, eksceso ir asimetrijos koeficiento reikšmėmis pagrįsti požymiai), tiek požymius, gautus pasitelkiant CNN.

Aritmijų klasifikavimui be prieš tai paminėtų metodų taip pat naudojami sprendimų medžiais [AA14] (angl. decision trees), giliuoju mokymusi [EX21] (angl. deep learning) paremti klasifikavimo metodai.

Remiantis apžvelgta literatūra ir atsižvelgiant į tai, kad tyrime siekiama išskirti požymius iš individualių kintamo ilgio širdies pūpsnių neturint charakteristinių taškų anotacijų, pasirinkta toliau tirti požymių išskyrimo metodus pagrįstus statistinėmis ir morfologinėmis širdies pūpsnių savybėmis, RR intervalais, taip pat požymių išskyrimą naudojant bangelių transformaciją, Ermito polinomų koeficientus.

Apžvelgtoje literatūroje dominuoja klasifikavimas naudojant SVM klasifikatorių. Šį klasifikavimo metodą galima laikyti kompromisu tarp paprastesnių (tokių kaip logistinė regresija ar tiesinio diskriminanto klasifikatorius), bet prastesnius rezultatus pasiekiančių, ir sudėtingų giliaisiais neuroniais tinklais paremtų klasifikatorių. Dėl šių priežasčių SVM klasifikatorius pasirinktas kaip perspektyvus naudoti šiame tyrime.

2. Duomenys ir metodologija

2.1. Duomenys

2.1.1. MIT-BIH duomenų bazė

Klasifikavimo modeliams mokyti ir įvertinti naudota individualių širdies pūpsnių duomenų aibė, sudaryta iš Massachusetts Institute of Technology-Beth Israel Hospital (toliau - MIT-BIH) aritmijų duomenų bazės duomenų.

MIT-BIH duomenų bazė yra pirmoji viešai prieinama aritmijų duomenų bazė, kuria naudojantis atlikta didžioji dalis publikuotų tyrimų, susijusių su automatiniu aritmijų aptikimu [dSSC+16]. Dėl šios priežasties ši duomenų bazė laikoma patikimiausiu standartiniu duomenų rinkiniu požymių išskyrimo iš elektrokardiogramos ir aritmijų klasifikavimo metodų tyrimui. MIT-BIH duomenų bazę sudaro iš 47 pacientų gautos 48 pusės valandos ilgio elektrokardiogramos ištraukos, įrašytos 1975-1979 metais BIH aritmijų laboratorijoje [MM01]. 23 įrašai parinkti naudojant paprastąją atsitiktinę imtį, tuo tarpu likusieji 25 parinkti siekiant į duomenų bazę įtraukti retesnes, bet kliniškai svarbias aritmijų rūšis, kurios nepakliūtų į atsitiktinę imtį. Šie signalai sudiskretitinti 360 Hz dažniu (laiko tarpas tarp dviejų gretimų taškų duomenų bazėje yra lygus 1/360 sekundės). Duomenų bazę iš viso sudaro duomenys apie daugiau nei 110 tūkstančių širdies pūpsnių, kurie anotuoti dviejų kardiologų sprendimu, visus kilusius nesutarimus išsprendus tarpusavyje [MM01].

2.1.2. Aritmijų klasės

Laikantis Association for the Advancement of Medical Instrumentation (toliau - AAMI) rekomendacijų aritmijų klasifikavimo sistemoms [dSSC+16], tyrime naudotos standartinės AAMI aritmijų klasės. MIT-BIH duomenų bazės anotacijose esančios aritmijų klasės ir jų atitikmenys AAMI klasėse pavaizduoti 1 lentelėje.

1 lentelė. AAMI aritmijų klasių standartai ir MIT-BIH aritmijų klasės

AAMI klasė	MIT-BIH klasė
Normal (N)	N, L, R, e, j
Supraventricular ectopic beat (S)	A, a, J, S
Ventricular ectopic beat (V)	V, E
Fusion beat (F)	F
Unknown beat (Q)	/, f, Q

Naudojant šias klases 90% širdies pūpsnių, esančių MIT-BIH duomenų bazėje, priklauso N klasei, 3% - SVEB , 6% - VEB, 1% - F. Q klasė duomenyse praktiškai neegzistuoja [MNR+19]. Šiame darbe tiriamas trijų didžiausių klasių (N, SVEB ir VEB) automatinis aptikimas.

2.2. Metodologija

2.2.1. Vertinimo schemos

Aritmijų klasifikavimo modeliams egzistuoja dvi pagrindinės duomenų dalijimo į mokymo ir testavimo aibes schemos [dSSC+16]:

- Pirmojoje (intrapacientinėje) dalijimo schemoje visų pacientų elektrokardiogramos duomenys sujungiami į bendrą duomenų rinkinį ir širdies pūpsniai atsitiktinai padalijami į mokymo ir testavimo aibes. Naudojant šį metodą modelis mokymo metu gali išmokti specifines pacientų detales, todėl naudojant testavimo aibę tikėtina gauti geresnį modelio kokybės įvertinimą negu galima tikėtis generalizuojant modelį (t.y. pritaikant modelį tokioje situacijoje, kai modeliui apmokyti neturima prieinamų ankstesnių duomenų apie pacientą) [COR04; LFD+11].
- Antrojoje (interpacientinėje) dalijimo schemoje, pasiūlytoje Chazal et al. [COR04], kiekvieno paciento širdies pūpsniai gali patekti tik į vieną iš mokymo arba testavimo aibių. Ši dalijimo schema laikoma realią širdies pūpsnių klasifikavimo situaciją atitinkančiu būdu įvertinti klasifikavimo metodų kokybę [EX21].

Atliktiems eksperimentams taikyta antroji (interpacientinė) schema.

2.2.2. Duomenų apdorojimas

Filtrai

Filtravimas yra procesas, kuriuo metu iš signalo pašalinamas nenorimos komponentes ar savybės. Žemų dažnių filtras (angl. low-pass filter) praleidžia tik žemo dažnio signalus. Aukštų dažnių (angl. high-pass filter), juostiniai (angl. band-pass) ir užtveriamieji (angl. band-stop) filtrai atitinkamai praleidžia aukšto dažnio, juostose tarp dviejų nustatytų ribų esančius ir visus į užtveriamąją juostą nepatenkančius dažnius.

Nukirtimo dažnis yra toks dažnis, už kurį didesnio dažnio signalai nėra praleidžiami filtro.

FIR fitrai

FIR filtrai yra filtrų tipas, kurių atsakas į baigtinės trukmės įvesties signalą yra baigtinės trukmės [RA21].

Diskretaus laiko N-tosios eilės FIR filtras yra svorinė paskutinių N+1 įvesties reikšmių svorinė suma (2.1).

$$y(n) = \sum_{i=0}^{N} b_i x(n-i), \tag{2.1}$$

kur:

- $x(n)$ - įvesties signalas
- $y(n)$ - gaunamas signalas
- N - filtro eilė
- b_i - i-tasis FIR filtro koeficientas

Ši transformacija bendru atveju dar vadinama signalo diskrečia konvoliucija.

FIR filtrai konstruojami randant tokius koeficientus b_i ir eilę, kad filtro paveiktas signalas atitiktų tam tikrą norimą specifikaciją.

Medianos filtras

Medianos filtras yra netiesinis filtras, naudojamas iš signalų pašalinti triukšmą. Naudojant medianos filtrą kiekvieno taško reikšmė yra pakeičiama tam tikro dydžio lango, centruoto aplink tą tašką, medianos reikšme. Kitaip negu tiesiniai filtrai, medianos filtrai gali eliminuoti nereprezentatyvių taškų, turinčių stipriai didesnes ar mažesnes reikšmes už kitus taškus lange, įtaką.

2.2.3. Požymių išskyrimas

Požymių išskyrimas yra esminis žingsnis automatiniam aritmijų aptikimui naudojant elektrokardiogramos signalą [dSSC+16]. Požymiais vadinama visa informacija gauta iš širdies pūpsnio signalo, kuri gali būti naudojama atskirti pūpsnio tipą.

Ermito polinomų koeficientai

Ermito polinomai yra apibrėžti rekurentiniu sąryšiu (2.2) [LFD+11].

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), \tag{2.2}$$

kur:

- $H_0 = 1$
- $H_1(x) = 2x$

Signalas gali būti reprezentuojamas kaip tiesinė Ermito polinomų iki eilės n kombinacija (2.3).

$$p(x) = c_0 + c_1H_1(x) + \cdots + c_nH_n(x). \tag{2.3}$$

Ermito polinomų koeficientai c_i gaunami minimizuojant (2.4).

$$\sum_j (y_j - p(x_j))^2, \tag{2.4}$$

kur:

- y_j - signalo reikšmė j-tajame taške
- x_j - j-tasis signalo taškas

Kuo didesnė Ermito polinomo eilė, tuo didesnis galimas jo kitimo dažnis laiko srityje ir geresnis gebėjimas rekonstruoti greitus pasikeitimus elektrokardiogramos signale. Įvertinti koeficientai gali būti naudojami kaip požymių vektorius, apibendrinantis širdies pūpsnio formą.

Aukštesnės eilės statistikos

Aukštesnės eilės statistikos (angl. higher-order statistics) yra imties funkcijos, kuriose naudojamas kėlimas trečiu arba aukštesniu laipsniu. Aukštesnės eilės statistikos naudojamos skaičiuojant formą apibūdinančius dydžius, tokius kaip ekscesas (2.5) ar asimetrijos koeficientas (2.6).

$$E\Big[\Big(\frac{X-\mu}{\sigma}\Big)^3\Big] = \frac{\mu^3}{\sigma^3}, \tag{2.5}$$

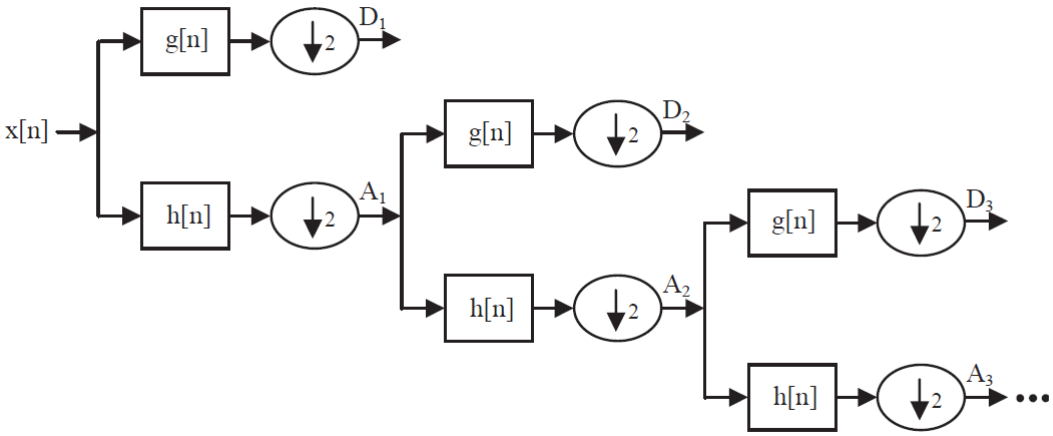
$$E\Big[\Big(\frac{X-\mu}{\sigma}\Big)^4\Big] = \frac{\mu^4}{\sigma^4}, \tag{2.6}$$

kur:

- μ^i - i-tasis centrinis momentas
- σ - standartinis nuokrypis

Banglių transformacija

Bangelių (taip pat vadinama vilnelių) transformacija (angl. wavelet transform) leidžia nagrinėti signalą tiek laiko, tiek dažnio srityse. Diskreti bangelių transformacija išskaido signalą į skirtingas skales, iš kurių kiekviena atitinka skirtingą dažnių juostą. Ši signalo dekompozicija gaunama pradiniam signalui atliekant konvoliucijos operaciją naudojant specialiai sudarytus filtrus (gaunamus iš pasirinktos motininės bangelės). Bangelių dekompozicijos schemoje (2 pav.) matoma, kad šis transformacijos metodas naudoja aukštų dažnių filtrą $g[n]$ išskirti pirmajam požymiui D_n. Žemų dažnių filtras $h[n]$ yra naudojamas gauti likusiai signalo daliai, iš kurios kartojant prieš tai aprašytą procedūrą išgaunami tolimesni požymiai.



2 pav. Bangelių transformacijos panaudojimas signalui išskaidyti [İE05]

Bangelių transformacijos metodą naudojant analizuoti signalus svarbus yra tinkamas motininės funkcijos ir dekompozicijos lygmenų parinkimas [dSSC+16].

2.2.4. Klasifikavimas

Iš širdies pūpsnių išskyrus pasirinktus požymius, sudaromi statistiniai ar mašininio mokymo modeliai, naudojantys šiuos požymius aritmijų klasifikavimui [dSSC+16].

Daugelio klasių klasifikavimas

Daugelio klasių klasifikavimas yra klasifikavimo uždavinių rūšis, kurioje kiekvienam objektui priskiriama viena iš daugiau negu dviejų galimų klasių.

Dalis klasifikavimo metodų savaime leidžia atlikti klasifikavimą turint daugiau negu dvi klases (pavyzdžiui multinominė logistinė regresija), tačiau kiti klasifikavimo metodai yra binarieji (galintys objektą priskirti tik vienai iš dviejų galimų klasių).

Vienas iš būdų pritaikyti binariuosius klasifikavimo metodus daugelio klasių klasifikavimo uždaviniui yra transformuojant šį uždavinį į kelis binarinio klasifikavimo uždavinius.

Egzistuoja dvi schemos šiai transformacijai:

- vienas-prieš-vieną (angl. One-Versus-One, toliau - OVO)
- vienas-prieš-likusius (angl. One-Versus-All, toliau - OVA)

Pirmojoje schemoje kiekvienai klasei sudaromas atskiras klasifikatorius, kuriame tos klasės stebėjimai laikomi teigiama klase, o visi likusieji - neigiama. Kiekvienas iš šių klasifikatorių pateiktam stebėjimui turi grąžinti pasikliovimo savo sprendimu reikšmę. Galutiniu daugelio klasių klasifikavimo uždavinio sprendimu laikoma klasės žyma to klasifikatoriaus, kuris grąžina didžiausią pasikliovimo reikšmę.

Antrojoje schemoje apmokomi $\frac{K(K-1)}{2}$ (kur K - galimų klasių skaičius) binarieji klasifikatoriai, iš kurių kiekvienas atitinką skirtingą dviejų klasių kombinaciją. Šių klasifikatorių spendimai sujungiami taip gaunant galutinį sprendimą. Paprasčiausias būdas gauti šį spendimą yra naudojant daugumos balsavimo (angl. majority voting) metodą, kuriuo galutiniu spendimu pasirenkama ta klasė, kuri daugiausia kartų buvo pasirinkta binariųjų klasifikatorių.

Atraminių vektorių mašina

Tiesinis klasifikatorius gali būti užrašomas lygtimi $w^T X + b = 0$. Ši lygtis apibrėžia ir tiesiniu SVM metodu ieškomą skiriamąją hiperplokštumą [CV95].

Klasifikuojant siekiama, kad:

$$w^T X_i + b > 0, \quad \text{kai } Y_i = 1, \tag{2.7}$$

$$w^T X_i + b < 0, \quad \text{kai } Y_i = -1, \tag{2.8}$$

kur:

- X_i - i-tojo objekto požymiai
- Y_i - i-tojo objekto klasės žyma

Pilno tiesinio klasių atskiriamumo (separabilumo) atveju SVM metodu hiperplokštuma ieškoma optimizuojant kvadratinio programavimo uždavinį:

$$\min_{w,b} \frac{1}{2} w^T w, \tag{2.9}$$

$$\text{su sąlyga } Y_i(w^T X_i + b) \geq 1. \tag{2.10}$$

Esminis SVM metodo ypatumas yra netiesinis n-matės erdvės atvaizdavimas kitoje (didesnės dimensijos) erdvėje, leidžiantis joje suformuoti tiesinę skiriamąją hiperplokštumą. Jeigu visiškas tiesinis klasių atskyrimas negalimas SVM metodą galima užrašyti kaip tokį optimizavimo uždavinį:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{L} \xi_i, \tag{2.11}$$

$$\text{su sąlyga } Y_i(w^T \phi(X_i) + b) \geq 1 - \xi_i, \tag{2.12}$$

$$\text{ir } \xi_i \geq 0 \quad i = 1,2...L. \tag{2.13}$$

Šiuo atveju SVM metodu ieškomas sprendimas yra hiperplokštuma $w^T \phi(X) + b = 0$.

Funkcija $\phi = \phi(X)$ tiesinį klasifikatorių padaro netiesiniu. Ji naudojama siekiant iš netiesiškai atskiriamų taškų suformuoti kitą vektorinę erdvę, kurioje tiesinis atskyrimas įmanomas.

Pavyzdžiui, jei $X \in R^2$, tai $\phi(X)$ galima apibrėžti kaip $\phi(X) = \left(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2\right)$.

Šios funkcijos, naudojamos SVM metode, apibrėžiamos per branduolio funkcijas $K(X,Y)$ naudojantis lygybe $K(X,Y) = \phi(X)\phi(Y)$, taip gaunant skaičiavimams naudingas savybes. Populiariausios branduolio funkcijos, naudojamos praktikoje, yra tiesinė, polinominė, radialinės bazės (angl. radial basis function).

Akivaizdu, kad kuo didesnė daugiklio C reikšmė, tuo labiau atsižvelgiama į blogiausiai mokymo aibėje klasifikuojamus duomenis, tačiau tuo sudėtingėja sprendimo paviršius ir mažėja modelio generalizacija (gebėjimas klasifikuoti prieš tai nematytus duomenis), todėl C galima laikyti SVM atvirkštiniu regularizacijos hiperparametru (modelio konfigūraciją nusakančiu parametru). Jei to reikalauja pasirinkta branduolio funkcija, taip pat turi būti parenkami ir kiti hiperparametrai.

SVM metodu gaunami geresni klasifikavimo rezultatai lyginant su kitais panašiais metodais, tačiau geram metodo veikimui reikalingas tinkamas hiperparametrų parinkimas. Šio metodo rezultatai yra jautrūs požymių matavimo skalėms, todėl būtinas požymių matavimo skalių suvienodinimas.

SVM yra binarusis klasifikavimo metodas, tačiau šį klasifikatorių galima pritaikyti daugelio klasių klasifikavimo uždaviniui naudojantis prieš tai aprašytomis uždavinio modifikavimo schemomis.

2.2.5. Modelio kokybės matavimas

Klasifikavimo tikslumas yra apibrėžiamas tikro teigiamo (angl. true positive, toliau - TP), tikro neigiamo (angl. true negative, toliau - TN), klaidingo teigiamo (angl. false positive, toliau - FP) ir klaidingo neigiamo (angl. false negative, toliau - FN) sąvokomis.

Automatinio aritmijų aptikimo modelių kokybės vertinimo būdai, rekomenduojami AAMI, yra jautrumas (angl. sensitivity, toliau - Se) (2.14), teigiamas prognostinis dydis (angl. positive predictivity arba precision, toliau - +P) (2.15), klaidingo teigiamo dažnis (angl. false positive rate, toliau - FPR) (2.16), tikslumas (angl. accuracy, toliau - Acc) (2.17) [dSSC+16].

$$Se = \frac{TP}{TP + FN}, \tag{2.14}$$

$$+P = \frac{TP}{TP + FP}, \tag{2.15}$$

$$FPR = \frac{FP}{TN + FP}, \tag{2.16}$$

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.17}$$

kur:

- TP - klasifikatoriumi gautas klasės buvimas jai esant
- TN - klasifikatoriumi gautas klasės nebuvimas jai nesant
- FP - klasifikatoriumi gautas klasės buvimas jai nesant
- FN - klasifikatoriumi gautas klasės nebuvimas jei esant

Daugelio klasių uždavinio atveju pirmosios trys charakteristikos gali būti skaičiuojamos kiekvienai klasei atskirai naudojant OVA schemą (t.y. visas likusias klases laikant neigiamomis). Tikslumo reikšmės kiekvienai klasei gaunamos identiškos, todėl prasminga kalbėti tik apie bendrą tikslumą. Stipriai išbalansuotų klasių atveju bendras tikslumas gali būti stipriai iškreiptas daugumos stebėjimų klasės rezultatų (širdies pūpsnių klasifikavimo atveju ši klasė yra normalūs širdies pūpsniai), todėl pirmieji trys iš anksčiau minėtų modelio tikslumo vertinimo būdų yra laikomi svarbiausiais tarpusavyje lyginant automatinio aritmijų aptikimo modelius [dSSC+16].

3. Pirminė duomenų analizė

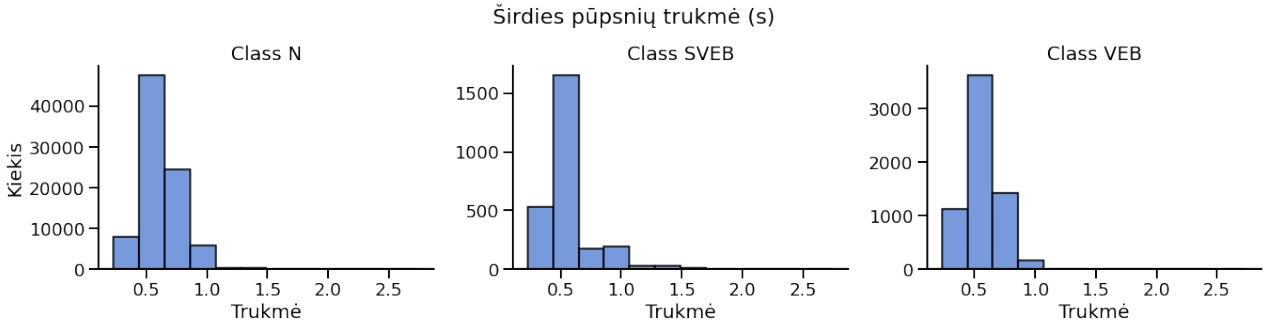
3.1. Aprašomoji statistika

Duomenų aibę sudarantys objektai yra iš anksto segmentuoti individualūs širdies pūpsniai. Apskaičiuotas širdies pūpsnių pasiskirstymas pagal klasę mokymo ir testavimo aibėse naudojant interpacientinę schemą (2 lentelė). Klasių pasiskirstymas stipriai išbalansuotas, daugumos klasę abiejuose aibėse sudaro sveiki širdies pūpsniai. Dėl šios priežasties pasirinkta laikyti, kad sudaromas klasifikatorius turi atsižvelgti į klasių išbalansavimą.

2 lentelė. Klasių pasiskirstymas naudojant interpacientinę schemą

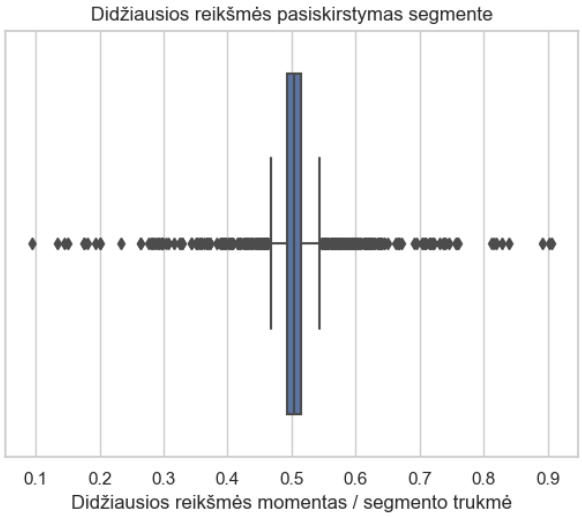
Klasė	Skaičius mokymo aibėje	Skaičius testavimo aibėje
N	43660	43214
SVEB	1813	818
VEB	3128	3204

Duomenų aibę sudaro nevienodo ilgio širdies pūpsniai. Pūpsnių ilgio pasiskirstymas kiekvienai klasei pavaizduotas histograma (3 pav.). Atsižvelgiant į kintamą pūpsnių trukmę, požymių išskyrimo metodai turi būti pasirinkti tokie, kurie geba gauti prasmingą informaciją iš kintamo dydžio širdies pūpsnių.



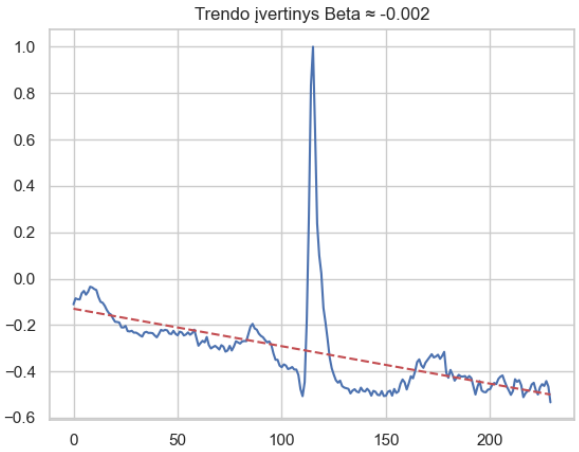
3 pav. Širdies pūpsnių trukmės pasiskirstymas pagal aritmijų klasę

Pūpsnių maksimumo taškų pasiskirstymas (4 pav.) parodo maksimumo taškų tendenciją būti arti pūpsnio vidurio. Pūpsnio maksimumo taškas galimai yra kartu ir R taškas, tačiau pūpsniuose turinčiuose stiprias maksimumo momento ir pūpsnio trukmės santykio išskirtis (*santykis* < 0.2 arba $0.8 <$ *santykis*) šie taškai dažniausiai skiriasi. Dėl šių priežasčių prasminga naudoti algoritminį R taško aptikimą.

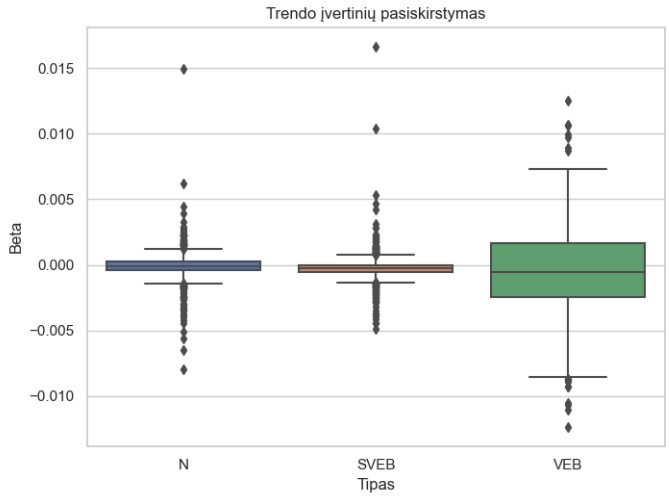


4 pav. Didžiausios reikšmės momento ir pūpsnio trukmės santykis. Imtis 1000 pūpsnių

Duomenų aibę sudarančiuose pūpsniuose nėra vienodos pradinės linijos (angl. baseline). Širdies pūpsnių laiko eilutėms pritaikius mažiausių kvadratų įvertį gautos paprastosios tiesinės regresijos kreivės (5 pav.). Šių kreivių krypties koeficientas beta β panaudotas kaip apytikslis trendo matas. Šių trendo reikšmių pasiskirstymas nėra vienodas visoje duomenų aibėje (6 pav.). Kadangi tokio tipo nukrypimai elektrokardiogramos signale gali būti susiję su išoriniais veiksniais kaip kvėpavimas ar blogas elektrodų kontaktas, pasirinkta laikyti, kad pradinės linijos nukrypimas gali neigiamai paveikti aritmijų aptikimą, todėl prasminga šį nukrypimą pašalinti.



5 pav. Nedidelio trendo įtaka tiesei



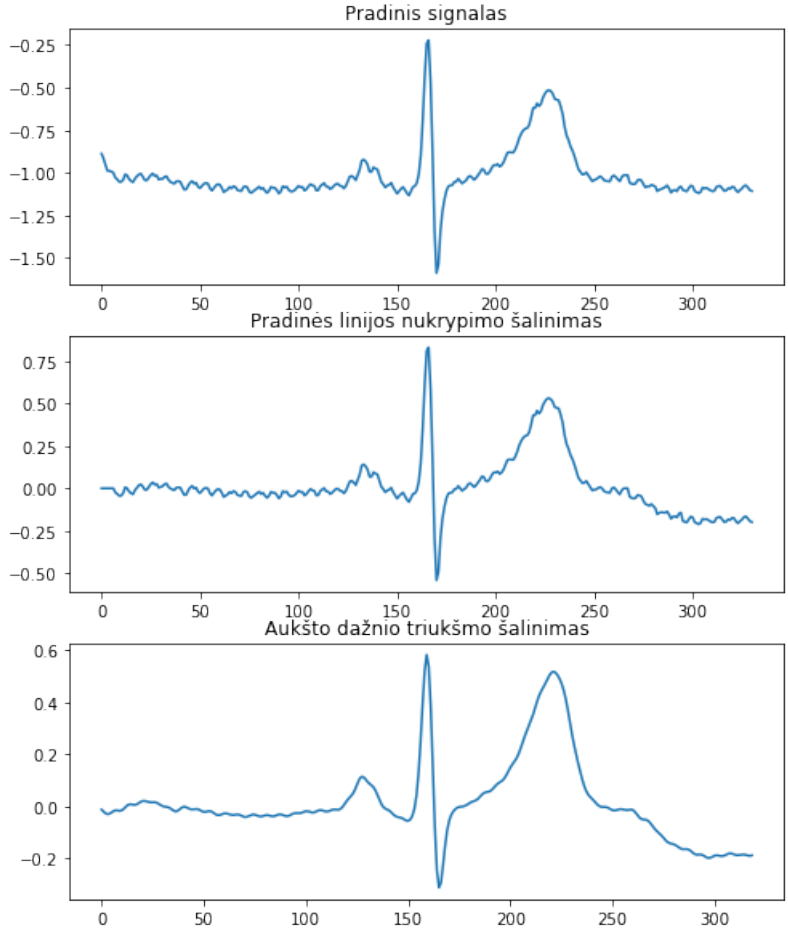
6 pav. Trendo įvertinių pasiskirstymas pagal aritmijų tipą

3.2. Pradinis apdorojimas

Visi toliau paminėti pradinio apdorojimo žingsniai atlikti tiek mokymo, tiek testavimo aibėms.

Remiantis dažna praktika, aprašyta [dSSC+16], pradinės linijos nukrypimui (angl. baseline wandering) pašalinti paeiliui pritaikyti 200 ir 600 ms pločio medianos filtrai, kurie panaikina atitinkamai QRS kompleksą/P bangas ir T bangas. Po filtravimo gautas rezultatas atimtas iš pradinio signalo. Aukšto dažnio triukšmas iš signalo pašalintas naudojant 12-eilės žemo dažnio FIR filtrą su 35 Hz nukirtimo reikšme.

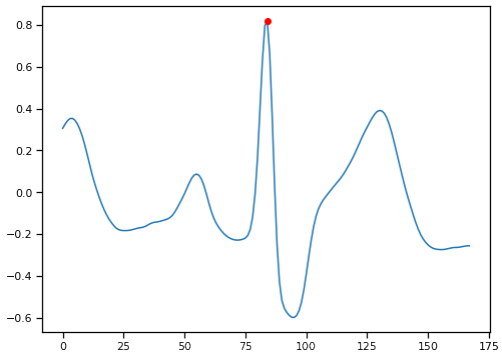
Atlikto apdorojimo rezultatų pavyzdys pavaizduotas grafiškai (7 pav.).



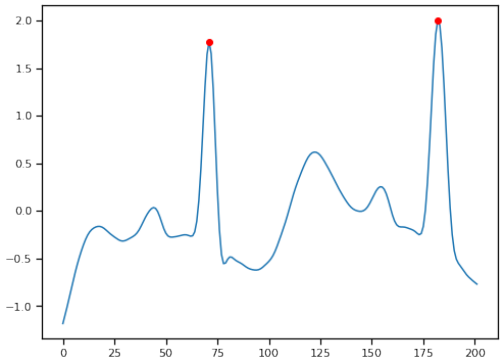
7 pav. Elektrokardiogramos pradinis apdorojimas

R taškai aptikti naudojant Christov [Chr04] algoritmo implementaciją. Algoritmo pritaikymas leidžia R taškus aptikti ir širdies pūpsniuose esant neįprastoms situacijoms (8 ir 9 pav.).

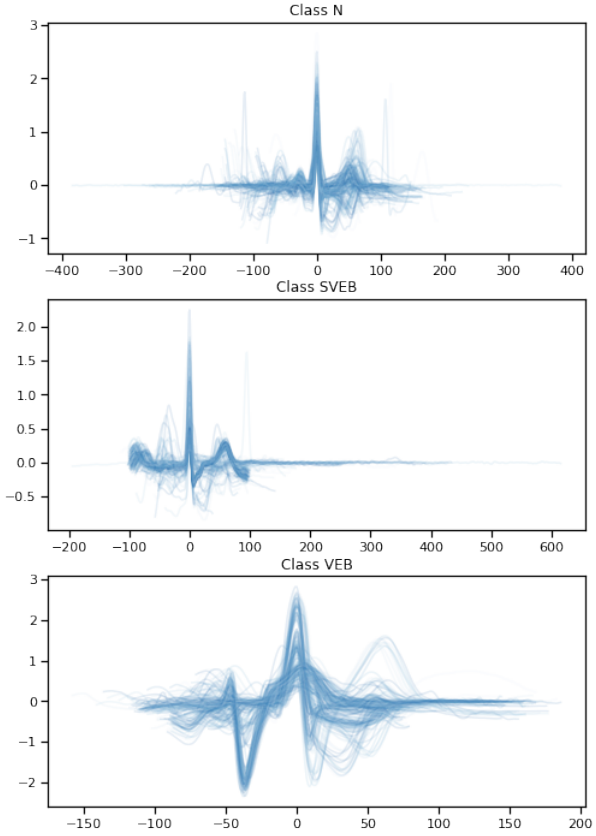
Siekiant vizualizuoti skirtumus tarp tiriamų aritmijų klasių, širdies pūpsnių imtis centruota aplink ją sudarančių pūpsnių R taškus ir šie pūpsniai nubraižyti viename grafike taip leidžiant kiekvienai klasei gauti tipinio širdies pūpsnio vaizdą (10 pav.).



8 pav. Įprastas širdies pūpsnys



9 pav. Neįprastas širdies pūpsnys



10 pav. Tipiniai pūpsniai kiekvienai aritmijų klasei

Stiprios maksimumo momento ir trukmės santykio išskirtys bei Christov algoritmu nustatyti neįprasti atvejai sudaro nestandartinių pūpsnių aibę. Ši aibė pasižymi nenatūralia pūpsnių išvaizda. Šie pūpsniai sudaro $\sim$ 1.2% mokymo aibės ir $\sim$ 1.6% testavo aibės. Nors duomenų ir klasės žymų teisingumas buvo užtikrintas medicinos specialistų, tačiau darant prielaidą, kad pūpsnius iš elektrokardiogramos signalo segmentuojantis algoritmas galėjo padaryti klaidų, aritmijų aptikimą esant nestandartiniams pūpsnių atvejams reikia analizuoti papildomai.

4. Modelio sudarymas ir eksperimentinis tyrimas

4.1. Požymių išskyrimas

Išskirti požymiai padalinti į grupes pagal tai, kokias metodais naudojantis jie gauti. Visų požymių matavimo skalės suvienodintos naudojant standartizavimą pagal vidurkį ir dispersiją (4.1). Požymių vidurkio ir dispersijos reikšmės gautos naudojantis mokymo duomenų aibe. Standartizuotų požymių vidutinės reikšmės kiekvienai aritmijų klasei pavaizduotos grafiškai (12 pav.).

$$z = \frac{x - \mu}{s}, \tag{4.1}$$

kur:

- z - standartizuota požymio reikšmė
- x - pradinė požymio reikšmė
- μ - požymio vidurkis mokymo aibėje
- s - požymio standatinis nuokrypis mokymo aibėje

Bangelių transformacija (1 grupė)

Remiantis [İE05], naudota Daubechies 2 eilės motininė bangelė ir signalas išskaidytas į 4 dekompozicijos lygmenis.

Kadangi turimi skirtingo ilgio širdies pūpsniai bei remiantis požymiais, naudotais [İE05; MML+11], kiekvienam dekompozicijos lygmeniui apskaičiuoti tokie jį apibendrinantys dydžiai:

- Didžiausia reikšmė.
- Mažiausia reikšmė.
- Didžiausios ir mažiausios reikšmės santykis.
- Vidurkis.
- Absoliučių reikšmių vidurkis.
- Dispersija.
- Asimetrijos koeficientas.

Taip požymių grupėje iš viso gaunant 35 požymius.

Ermito polinomų koeficientai (2 grupė)

Ermito polinomų koeficientai apskaičiuoti atskirai imant Ermito polinomus iki 3, 4 ir 5 laipsnių ir taip gaunant atitinkamai po 4, 5 ir 6 koeficientus. Šie požymiai sujungiami taip šioje požymių grupėje iš viso gaunant 15 požymių.

Aukštesnės eilės statistikos (3 grupė)

Remiantis [MNR+19], pasirinkta apskaičiuoti eksceso ir asimetrijos koeficiento reikšmes skirtingiems pūpsnio intervalams. Atsižvelgiant į tai, kad turimi skirtingo ilgio širdies pūpsniai, kiekvienas iš jų padalintas į 6 intervalus, gautus abi puses nuo R taško padalijus į 3 lygius intervalus.

Morfologiniai ir statistiniai požymiai (4 grupė)

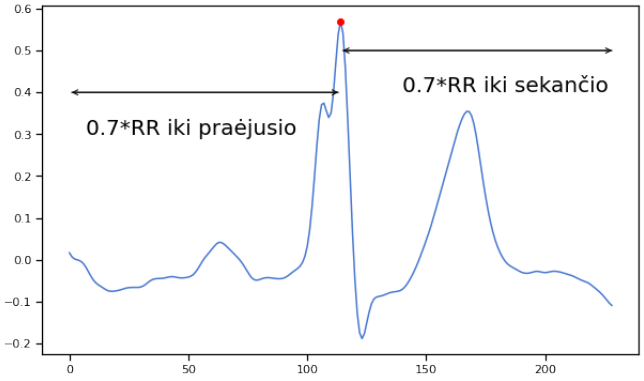
Remiantis [MML+11] naudotais požymiais, išskirta 11 morfologinių ir statistinių požymių. Išskirti morfologiniai požymiai:

- Mažiausia pūpsnio reikšmė.
- Didžiausia reikšmė.
- Didžiausios ir mažiausios reikšmės santykis.
- Teigiamų reikšmių plotas.
- Neigiamų reikšmių plotas.

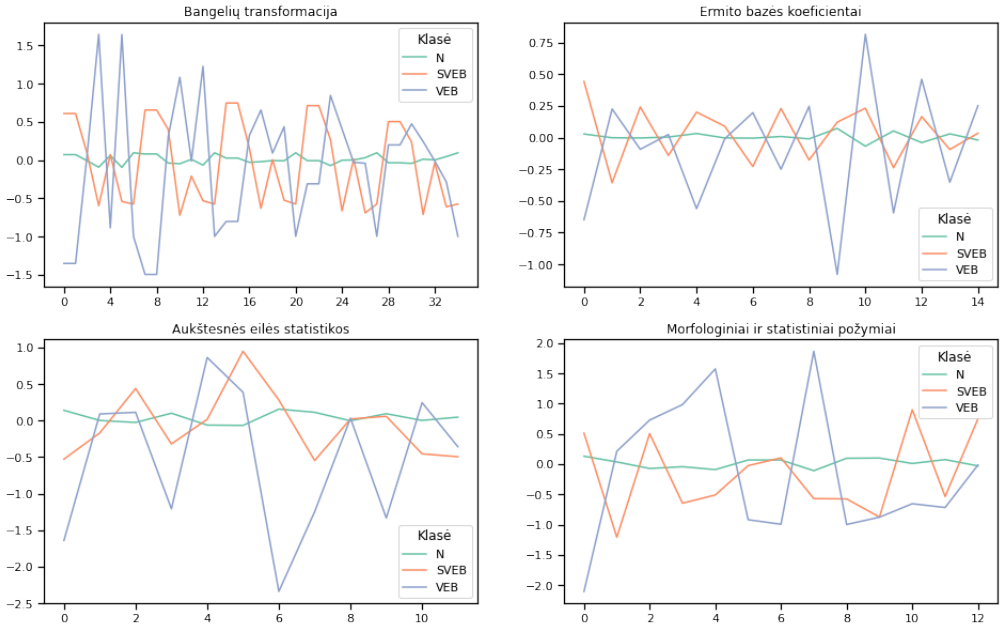
Išskirti statistiniai požymiai:

- Amplitudės vidurkis.
- Mediana.
- Dispersija.
- Asimetrijos koeficientas.
- Ekscesas.
- Histogramos dispersija (dalijant į 20 dalių).

Žinoma, kad kiekvienas duomenyse turimas širdies pūpsnys buvo segmentuotas paliekant tam tikrą atstumą nuo R taško į abi puses, kuris priklauso nuo praėjusio ir sekančio pūpsnio R taško (11 pav.). Naudojant šią schemą, suskaičiuoti RR intervalai iki praėjusio ir sekančio pūpsnio, taip su šiais dviem papildomais požymiais iš viso grupėje gaunant 13 požymių.



11 pav. RR intervalų apskaičiavimo schema



12 pav. Vidutinės standartizuotų požymių reikšmės kiekvienai aritmijų klasei (požymiai išrikiuoti pagal jų aprašymo tvarką numeraciją pradedant nuo 0)

4.2. Klasifikavimas

Klasifikavimui pasirinkta naudoti logistinės regresijos ir SVM metodus. Iš šių dviejų metodų logistinė regresija yra konceptualiai paprastesnis, lengviau interpretuojamas ir greitesniu klasifikatoriaus mokymo laiku pasižymintis metodas, tuo tarpu naudojant SVM metodą tikimasi gauti geresnius klasifikavimo rezultatus.

Visi sudaryti modeliai apmokyti naudojant mokymo aibę, modelių kokybė įvertinta naudojant testavimo aibę.

Logistinės regresijos modeliai sudaryti nenaudojant jokio regularizacijos metodo, naudojant stebėjimų kiekiui proporcingus klasių svorius, apskaičiuojamus pagal (4.2):

$$w_c = \frac{n}{c \cdot n_c}, \tag{4.2}$$

kur:

- w_c - klasės c svoris
- n - bendras stebėjimų skaičius
- c - klasių skaičius
- n_c - klasės c stebėjimų skaičius

Remiantis [MNR+19], atraminių vektorių mašinos modelius pasirinkta sudaryti naudojant radialinės bazės branduolio funkciją. Optimalios parametrų C ir γ reikšmėms surastos naudojantis kryžmine validacija (angl. cross-validation) su logaritminiu parametrų reikšmių tinkleliu. Remiantis [MNR+19], ieškant geriausio parametrų rinkinio papildomai patikrinta γ reikšmė, proporcinga naudojamam požymių skaičiui ($\gamma = \frac{1}{\text{požymių skaičius}}$). Klasių išbalansavimo problemai spręsti naudota svorių schema aprašyta viršuje. Daugelių klasių uždaviniui SVM klasifikatorius pritaikytas naudojant OVO schemą.

Norint gauti galutinį klasifikatoriaus sprendimą naudotos dvi balsavimo schemos: paprasta, pasirenkanti tą klasę, kurią pasirinko dauguma OVO klasifikatorių, ir sudėtingesnė, kiekvienai klasei iš OVO klasifikatorių grąžintų reikšmių sudaranti bendrą balą (4.3) ir galutiniu spendimu pasirenkanti tą klasę, kurios bendras balas didžiausias (4.4):

$$\delta_n = \sum_{l \in L} y_n f_l(x), \tag{4.3}$$

kur:

- x - klasifikuojamo objekto požymiai
- L - OVO klasifikatorių aibė
- f_l - l-tojo OVO klasifikatoriaus grąžinama reikšmė
- y_n - $\begin{cases} 1 & \text{jei OVO klasifikatoriumi gauta } n\text{-toji klasė} \\ -1 & \text{jei OVO klasifikatoriumi gauta ne } n\text{-toji klasė} \\ 0 & \text{jei OVO klasifikatorius neklasifikuoja } n\text{-tosios klasės} \end{cases}$

$$\text{argmax}_n \delta_n. \tag{4.4}$$

Tokią (antrąją) galutinio sprendimo schemą galima interpretuoti kaip balsavimą, kuriame stipriai savo sprendimu pasikliaujančių klasifikatorių balsams suteikiamas didesnis svoris. Naudojant pirmą balsavimo schemą lygiųjų atveju klasifikatoriaus galutiniu sprendimu pasirenkama pirmoji (normalių pūpsnių) klasė. Laikyta, kad turint tik 3 OVO modelius, šį rezultatą gauti yra pakankamai tikėtina, todėl klasifikatoriumi gaunami rezultatai gali tapti per daug konservatyvūs (pūpsniams nepriskiriamos patologinės klasės). Kadangi naudojant antrąją balsavimo schemą lygiųjų yra išvengiama, tai motyvuoja prieš tai aprašytų balsavimo schemų palyginimą.

4.3. Atlikti eksperimentai

Lyginamasis klasifikatorius

Siekiant įvertinti išskirtų požymių gebėjimą klasifikuoti aritmijų klases, sudaryti lyginamieji (angl. baseline) logistinės regresijos ir SVM modeliai, kuriems vietoje išskirtų požymių pateikimas pats elektrokardiogramos signalas. SVM modeliui naudota paprastesnė (pirmoji) balsavimo schema ir naudoti pradiniai modelio parametrai.

Siekiant suvienodinti signalo ilgį pasirinkta imti 150 taškų dydžio langą, centruotą pūpsnio R taške (imant po 75 taškus į abi puses). Jeigu tokio dydžio lango paimti į kurią nors pusę nebuvo galima, signalas pratęstas paskutine turima reikšme. Lyginamuoju SVM modeliu gauti geresni rezultatai, negu naudojant logistinę regresiją (3 lentelė). Šioje ir tolimesnėse lentelėse pateikiamos testavimo aibėje (aprašytos 3.1 skyriuje) gautos modelio kokybės metrikų, aprašytų 2.2.5 skyriuje, reikšmės.

3 lentelė. Lyginamųjų klasifikatorių rezultatai

	N			SVEB			VEB			
Tipas	S	P+	FPR	S	P+	FPR	S	P+	FRP	Acc
Logistinė regresija	.755	.971	.242	.468	.08	.094	.573	.207	.16	.738
SVM	.797	.981	.165	.468	.084	.09	.781	.33	.115	.79

Požymių grupių įvertinimas

Abiejų tipų modeliai apmokyti naudojant po vieną požymių grupę, taip siekiant įvertinti požymių išskyrimo metodų gebėjimą atskirti aritmijų grupes. Taip pat palyginimui pateikiami rezultatai, gaunami kartu naudojant visas požymių grupes. Naudojant logistinę regresiją geriausi vienos požymių grupės rezultatai gauti naudojant morfologinių ir statistinių požymių grupę (4 grupė), tuo tarpu prasčiausi gauti su Ermito polinomų koeficientų grupe (2 grupė) (4 lentelė). Panašūs rezultatai gauti ir naudojant SVM klasifikatorių (5 lentelė). Rezultatuose papildomai pastebėtas visiškas bangelių transformacijos požymių grupės (1 grupė) negebėjimas aptikti SVEB aritmijų klasę. Lyginant dvi galutinio sprendimo schemas nerasta sistemingų skirtumų, leidžiančių vieną schemą įvardinti kaip geresnę. SVM klasifikatoriumi ryškiai pagerinamas logistine regresija gaunamas bendras tikslumas, tačiau šis pagerėjimas susijęs su tikslesniu normalių širdies pūpsnių klasifikavimu. Patologinių klasių atpažinimo atžvilgiu SVM modelis konservatyvesnis už logistinę regresiją: lyginant tuos pačius požymių rinkinius naudojančius modelius, SVM metodas nėra linkęs pūpsnius klasifikuoti kaip patologinius jiems tokiais nesant, tačiau dėl to nukenčia bendras patologinių pūpsnių aptikimas. Abiems modeliams aptinkant VEB patologinę klasę gauti geresni rezultatai lyginant su SVEB klasės aptikimu.

4 lentelė. Rezultatai skirtingoms požymių grupėms naudojant logistinės regresijos klasifikatorių

	N			SVEB			VEB			
Požymių grupė	S	P+	FPR	S	P+	FPR	S	P+	FRP	Acc
1	.599	.973	.181	.611	.049	.207	.825	.251	.179	.614
2	.596	.938	.420	.306	.024	.215	.459	.154	.083	.582
3	.694	.982	.135	.438	.034	.216	.831	.423	.083	.699
4	.736	.982	.143	.626	.055	.188	.831	.470	.067	.740
Visos	.687	.986	.116	.604	.063	.159	.874	.304	.146	.699

5 lentelė. Rezultatai skirtingoms požymių grupėms naudojant SVM klasifikatorių

		N			SVEB			VEB			
Sprendimo schema	Požymių grupė	S	P+	FPR	S	P+	FPR	S	P+	FRP	Acc
1	1	.830	.952	.452	.044	.017	.144	.643	.275	.123	.804
2	1	.923	.940	.638	.044	.025	.031	.422	.409	.044	.874
1	2	.800	.949	.465	.312	.032	.166	.511	.585	.026	.772
2	2	.804	.940	.553	.336	.034	.169	.399	.593	.002	.769
1	3	.844	.968	.302	.268	.042	.108	.708	.518	.048	.825
2	3	.796	.963	.326	.278	.036	.131	.656	.404	.071	.778
1	4	.778	.970	.258	.517	.056	.154	.728	.464	.061	.770
2	4	.827	.960	.372	.502	.061	.137	.611	.608	.029	.807
1	Visos	.919	.975	.250	.381	.467	.008	.798	.437	.075	.901
2	Visos	.940	.967	.342	.380	.493	.007	.691	.479	.055	.913

Geriausias patologinių pūpsnių klasių aptikimas

Abiems modeliams siekta surasti požymių rinkinių kombinacijas, kuriomis naudojantis gaunamas geriausias patologinių aritmijų klasių aptikimas joms esant (t.y. gaunamos didžiausios modelio jautrumo reikšmės). Geriausi rezultatai pateikti atskirai SVEB ir VEB klasėms. Papildomai pateikti požymio rinkinio, su kuriuo gaunamas geriausias bendras tikslumas, rezultatai. Geriausias VEB ir SVEB klasių aptikimas naudojant logistinę regresiją gautas atitinkamai su 3,4 ir 1,2,4 požymių grupių kombinacijomis. Geriausias bendras tikslumas gautas kartu naudojant 2 ir 4 požymių grupes (6 lentelė). Visi šie modeliai pagerina lyginamuoju logistinės regresijos modeliu gautus rezultatus. Naudojant SVM metodą geriausias abiejų tirtų patologinių klasių aptikimas gautas naudojant vien tik 4 požymių grupę, geriausias bendras tikslumas - sujungiant 2,3 ir 4 grupes (7 lentelė). Geriausiai SVEB patologinę klasę aptinkantis modelis nepagerina šios klasės aptikimo rezultatų naudojant lyginamąjį modelį. Šis rezultatas vėl parodo SVM modelių konservatyvumo problemą - geriausio rasto modelio P+ reikšmė VEB klasei daugiau nei dvigubai didesnė už šią reikšmę lyginamajam modeliui.

6 lentelė. Požymių kombinacijos, su kuriomis gaunami geriausi rezultatai naudojant logistinės regresijos klasifikatorių (geriausi rezultatai paryškinti)

	N			SVEB			VEB			
Požymių grupė	S	P+	FPR	S	P+	FPR	S	P+	FRP	Acc
3,4	.734	.987	.105	.577	.062	.153	**.890**	.379	.106	.742
1,2,4	.688	.985	.113	**.671**	.073	.113	.878	.295	.153	.700
2,4	.792	.982	.016	.643	.084	.123	.811	.424	.008	**.791**

7 lentelė. Požymių kombinacijos, su kuriomis gaunami geriausi rezultatai naudojant SVM klasifikatorių (geriausi rezultatai paryškinti)

Sprendimo schema	Požymių grupė	N			SVEB			VEB			Acc
		S	P+	FPR	S	P+	FPR	S	P+	FRP	
1	4	.778	.970	.258	**.517**	.056	.154	**.728**	.464	.061	.770
2	2,3,4	.989	.946	.612	.164	.338	.006	.407	.797	.008	**.935**

Pradinio apdorojimo įtaka

Siekta įvertinti pradinės linijos nukrypimo ir aukšto dažnio triukšmo šalinimo (atitinkamai medianos ir FIR filtrais) įtaką klasifikavimo rezultatams. Visi požymiai pakartotinai išskirti naudojant tuos pačius metodus, tačiau prieš tai neatlikus pradinio elektrokardiogramos apdorojimo. Modeliai iš naujo apmokyti naudojant po vieną požymių grupę, taip leidžiant įvertinti kaip pradinis apdorojimas veikia kiekvienos požymių grupės klasifikavimo kokybę. Papildomai įvertinti rezultatų skirtumai modelyje naudojant visas požymių grupes. Paprastumo dėlei SVM modeliui naudota pirmoji galutinio sprendimo schema.

Logistinei regresijai gauti rezultatai nėra vienareikšmiški, tačiau pagal visas naudojamas modelio kokybės matavimo metrikas pastebėtas neryškus SVEB klasės aptikimo pagerėjimas atlikus pradinį signalo apdorojimą (8 lentelė). Naudojant SVM klasifikatorių sistemingo tam tikros aritmijų klasės aptikimo pagerėjimo nerasta (9 lentelė). Abiejų klasifikatorių atvejais pastebėta ryški teigiama pradinio apdorojimo įtaka bendram klasifikatoriaus tikslumui naudojant aukštesnės eilės statistikų požymių grupę (3 grupė).

8 lentelė. Rezultatų pagerėjimas atlikus pradinį apdorojimą naudojant logistinės regresijos klasifikatorių

Požymių grupė	N			SVEB			VEB			Acc
	S	P+	FPR	S	P+	FPR	S	P+	FRP	
1	-.040	-.002	.003	.093	.005	.010	-.035	-.044	.030	-.038
2	-.083	-.013	.048	.094	.007	.005	-0.111	-.126	.078	-.081
3	.121	.012	-.052	.156	.015	-0.42	.051	.172	-.086	.125
4	-.024	0.34	.020	.059	-.005	.034	-.026	.010	-0.23	-.024
Visos	.002	.003	-.013	.051	.001	.011	.014	.034	-.023	.014

9 lentelė. Rezultatų pagerėjimas atlikus pradinį apdorojimą naudojant SVM klasifikatorių

Požymių grupė	N			SVEB			VEB			Acc
	S	P+	FPR	S	P+	FPR	S	P+	FRP	
1	-.045	.007	-.122	.021	-.009	.029	.131	-.088	.058	-.071
2	-.009	-.001	.01	-.155	-.018	0.11	-.011	-.003	-.001	-.014
3	.115	.001	.045	-.111	.001	.048	-.034	.222	-.060	.119
4	-0.63	-0.18	.085	–0.70	-0.66	.079	-.160	.020	-.021	-.017
Visos	.017	.006	-.073	.161	.339	-.019	.034	-.102	.028	-.001

Pūpsnių išskirčių įtaka

Siekta papildomai ištirti pirminės duomenų analizės metu rastų pūpsnių išskirčių aibės klasifikavimą. Eksperimentui naudoti logistinės regresijos ir SVM modeliai, pasiekę didžiausią bendrą tikslumą, taip pat visas požymių grupes naudojantys modeliai. Paprastumo dėlei SVM modeliui naudota pirmoji galutinio sprendimo schema. Visais atvejais pastebėtas stipriai suprastėjęs patologinių klasių aptikimas (10 lentelė). Didžiausias skirtumas rastas naudojant geriausią bendrą tikslumą visoje testavimo pūpsnių aibėje pasiekusį SVM modelį - šis modelis beveik išvis negebėjo aptikti patologinių pūpsnių, esančių išskirčių aibėje. Daryta išvada, kad reikalinga tolesnė analizė, siekiant nustatyti kurios požymių grupės (ar individualūs požymiai) praranda gebėjimą aptikti patologinius pūpsnius turint pūpsnius, priklausančius išskirčių aibei.

10 lentelė. Pūpsnių išskirčių klasifikavimas naudojant logistinę regresiją ir SVM

Tipas	Požymių grupė	N			SVEB			VEB			Acc
		S	P+	FPR	S	P+	FPR	S	P+	FRP	
Logistinė regresija	2,4	.764	.938	.108	.571	.044	.115	.807	.767	.115	.775
Logistinė regresija	Visos	.812	.895	.204	.143	.014	.094	.747	.806	.081	.786
SVM	2,3,4	.998	.683	.988	.000	.000	.000	.009	.500	-.004	.682
SVM	Visi	.885	.839	.362	.286	.037	.007	.609	.899	.031	.794

Rezultatai ir išvados

Nagrinėtoje literatūroje požymiai daugiausiai išskirti remiantis pūpsnių charakteristiniais taškais, RR intervalais, morfologinėmis ir statistinėmis savybėmis, dimensijos mažinimo metodais. Norint išskirti nemažą dalį šių požymių būtina turėti papildomą informaciją, pavyzdžiui, iš anksto turimas charakteristinių taškų vietos anotacijas ar duomenis apie iš eilės sekančius vieno paciento širdies pūpsnius, kas nėra galima turint individualių širdies pūpsnių duomenis. Dalis kitų požymių taip pat negali būti išskirti dėl kintamo širdies pūpsnių ilgio. Apžvelgtuose straipsniuose dominavo SVM klasifikatoriaus taikymai.

Pirminė duomenų analizė parodė stiprų klasių išbalansavimą turimoje duomenų aibėje, taip pat ryškius pūpsnių trukmės skirtumus. Daryta išvada, kad sudaromas klasifikatorius turi atsižvelgti į klasių išbalansavimą, išskirti požymiai turi gauti prasmingus rezultatus kintamo ilgio širdies pūpsniams. Remiantis nagrinėta literatūra širdies pūpsniams atliktas pradinis apdorojimas naudojant FIR ir medianos filtrus.

Atsižvelgiant į nagrinėtą literatūrą ir pirminę duomenų analizę, požymius pasirinkta išskirti naudojantis bangelių transformacija, Ermito polinomų koeficientais, aukštesnės eilės statistikomis, morfologinėmis/statistinėmis pūpsnių savybėmis. Klasifikavimui atlikti pasirinkti naudoti ir tarpusavyje palyginti logistinės regresijos ir SVM metodai (antrajam metodui lyginant dvi skirtingas galutinio sprendimo schemas). Remiantis nagrinėta literatūra SVM naudota radialinės bazės branduolio funkcija. Optimalių SVM parametrų paieškai taikyta kryžminė validacija naudojant logaritminį parametrų reikšmių tinklelį.

Naudojant individualias išskirtų požymių grupes geriausi patologinių pūpsnių aptikimo rezultatai gauti naudojantis morfologinių/statistinių požymių grupe, prasčiausi - Ermito polinomų koeficientais. Naudojant požymių grupių kombinacijas abiejų klasifikavimo metodų atveju gauti modeliai, ryškiai pagerinantys lyginamųjų modelių rezultatus, išskyrus VEB patologinės klasės aptikimo jautrumą naudojant SVM metodą (.728 pagal šią metriką geriausiam išskirtus požymius naudojančiam modeliui, .781 lyginamajam modeliui). Aptinkant SVEB aritmijų klasę visais tirtais atvejais gauti prasčiausi rezultatai iš tyrime naudotų aritmijų klasių. Tolimesniuose tyrimuose prasminga sudaryti geriausiai klasifikuojančius požymių rinkinius išskaidant išskirtų požymių grupes į atskirus požymius (požymių vertinimui naudojant filtravimo ar aplanko metodus). Nesvarbių požymių atsisakymas galėtų pagerinti SVEB klasės aptikimo rezultatus.

Atraminių vektorių mašinos metodu gautas ryškiai geresnis klasifikavimo tikslumas (geriausias bendras tikslumas logistinės regresijos ir SVM modeliams atitinkamai .791 ir .935), bet patologinių širdies pūpsnių klasių aptikimo atžvilgiu rezultatai yra konservatyvūs lyginant su logistinės regresijos metodu. Prasminga toliau nagrinėti galimus šios problemos sprendimus, pavyzdžiui, padidinant patologinių klasių svorius arba modeliui apmokyti naudojant dirbtinų stebėjimų generavimo algoritmus (tokius kaip SMOTE). Naudojant SVM su svorine galutinio sprendimo priėmimo schema negauti geresni klasifikavimo rezultatai lyginant su daugumos balso galutinio sprendimo schema. Ateities darbuose naudinga toliau nagrinėti kitomis galutinio sprendimo schemomis gaunamus rezultatus, pavyzdžiui, naudojant nested algorithm schemą, aprašytą [BZX07].

Rasta, kad dažnai literatūroje taikomas pradinis elektrokardiogramos signalo apdorojimas FIR ir medianos filtrais neryškiai pagerina SVEB patologinės klasės aptikimą naudojant logistinės regresijos klasifikatorių (tarp tirtų atvejų didžiausias SVEB klasės jautrumo padidėjimas buvo lygus .156), tačiau naudojant SVM metodą sistemingai nepagerina jokių aritmijų klasių aptikimo. Atsižvelgiant į gautus rezultatus, vienas iš uždavinių tolesniuose tyrimuose galėtų būti sudėtingesnių pradinio apdorojimo ir triukšmo šalinimo metodų, tokių kaip bangelių transformacija, įtakos klasifikavimo rezultatams tyrimas.

Tyrime sudaryta metodologija, širdies pūpsnius priskirti išskirčių aibei. Naudojant sudarytą išskirčių aibę rasta, kad kai kurie prieš tai sudaryti klasifikatoriai nesugeba aptikti išskirčių aibėje esančių patologinių širdies pūpsnių (geriausią bendrą tikslumą su visų pūpsnių aibe pasiekusio SVM klasifikatoriaus jautrumo reikšmės SVEB ir VEB klasėms naudojant pūpsnių išskirčių aibę gautos lygios atitinkamai .000 ir .009). Prasminga toliau nagrinėti, kokios specifinės požymių grupės (ar individualūs požymiai) labiausiai praranda gebėjimą aptikti patologines aritmijų klases kai nagrinėjami išskirčių aibei priklausantys širdies pūpsniai.

Šaltiniai

[AA14] R. Ahmed ir S. Arafat. Cardiac arrhythmia classification using hierarchical classification model. 2014.

[ALP12] A.S. Alvarado, C. Lakshminarayan ir J.C. Principe. Time-Based Compression and Classification of Heartbeats. *IEEE Transactions on Biomedical Engineering*, 59(6):1641–1648, 2012.

[BZX07] L. Bo, H. Zhifeng ir Y. Xiowei. Nesting Algorithm for Multi-Classification Problem. *Soft Computing*, 11:383–389, 2007.

[Chr04] I. Christov. Real time elecrocardiogram QRS detection using combined adaptive threshold. *Biomedical engineering online*, 3:28, 2004.

[COR04] P. Chazal, M. O'Dwyer ir R. Reilly. Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features. 51:1196–206, 2004.

[CV95] C. Cortes ir V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[dSSC+16] E. José da S. Luz, W.R Schwartz, G. Cámara-Chávez ir D. Menotti. ECG-based heartbeat classification for arrhythmia detection: A survey. *Computer Methods and Programs in Biomedicine*, 127:144–164, 2016.

[EX21] E. Essa ir X. Xie. An Ensemble of Deep Learning-Based Multi-Model for ECG Heartbeats Arrhythmia Classification. *IEEE Access*, 9:103452–103464, 2021.

[GP12] A. Gacek ir W. Pedrycz. *ECG Signal Processing, Classification and Interpretation*. Springer-Verlag, Londonas, Jungtinė Karalystė, 2012. 172 psl.

[HLZ+14] H. Huang, J. Liu, Q. Zhu, R. Wang ir G. Hu. A new hierarchical method for inter-patient heartbeat classification using random projections and RR intervals. *Biomedical engineering online*, 13, 2014.

[HPA18] A. Hernández-Madrid, T. Paul ir D. Abrams. Arrhythmias in congenital heart disease: a position paper of the European Heart Rhythm Association (EHRA), Association for European Paediatric and Congenital Cardiology (AEPC), and the European Society of Cardiology (ESC) Working Group on Grown-up Congenital heart disease, endorsed by HRS, PACES, APHRS, and SOLAECE. *EP Europace*, 20(11):1719–1753, 2018.

[IBA19] I.Sadek, J. Biswas ir B. Abdulrazak. Ballistocardiogram signal processing: a review. *Health Information Science and Systems*, 7, 2019.

[YCK10] C. Ye, M. Coimbra ir B. Kumar. Arrhythmia Detection and Classification using Morphological and Dynamic Features of ECG Signals. *Conference proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*:1918–1921, 2010.

[İE05] G. İnan ir U.D. Elif. ECG beat classifier designed by combined neural network model. *Pattern Recognition*, 38(2):199–208, 2005.

[KSS+09] J. Kim, H. Shin, K. Shin ir M. Lee. Robust algorithm for arrhythmia classification in ECG using extreme learning machine. *Biomedical engineering online*, 8, 2009.

[LFD+11] G. Lannoy, D. François, J. Delbeke ir M. Verleysen. Weighted SVMs and Feature Relevance Assessment in Supervised Heart Beat Classification. *Commun Comput Inf Sci*, 127, 2011.

[MAL+13] R. Martis, U. Acharya, C. Lim ir J. Suri. Characterization of ECG beats from cardiac arrhythmia using discrete cosine transform in PCA framework. *Knowledge-Based Systems*, 45:76–82, 2013.

[MM01] G.B. Moody ir R.G. Mark. The impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[MML+11] T. Mar, S. Zaunsederand J. Martínez, M. Llamedo ir R. Poll. Optimization of ECG Classification by Means of Feature Selection. *Biomedical Engineering, IEEE Transactions on*, 58:2168–2177, 2011.

[MNR+19] V. Mondéjar-Guerra, J. Novo, J. Rouco, M.G. Penedo ir M. Ortega. Heartbeat classification fusing temporal and morphological information of ECGs via ensemble of classifiers. *Biomedical Signal Processing and Control*, 47:41–48, 2019.

[RA21] R. Subhabrata Roy ir C. Abhijit. A Survey of FIR Filter Design Techniques: Low-complexity, Narrow Transition-band and Variable Bandwidth. *Integration*, 77:193–204, 2021.

[RMJ+20] G.A Roth, G.A Mensah, C.O. Johnson ir G. Addolorato. Global Burden of Cardiovascular Diseases and Risk Factors, 1990–2019: Update From the GBD 2019 Study. *Journal of the American College of Cardiology*, 76(25):2982–3021, 2020.

[SH18] A. Sellami ir H. Hwang. A Robust Deep Convolutional Neural Network with Batch-Weighted Loss for Heartbeat Classification. *Expert Systems with Applications*, 122, 2018.

Priedai

1 Priedas. Pirminės analizės programinis kodas

```
### Pirmine analize

#### Vizualizavimas

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

df = pd.read_csv("train_data_real_signal_ft.csv")
x = df.copy(deep=True)
x_eval = pd.read_csv("test_data_real_signal_ft.csv")

x_small = x.sample(3000)

x_small_eval = x_eval.sample(3000)

x.groupby("label")["seq_size"].count()

x_eval.groupby("label")["seq_size"].count()

x_full = x.append(x_eval)
prop = x_full.groupby("label").count()["seq_size"]
prop / prop.sum()

x_numpy = x_small.iloc[:,2:].to_numpy()
x_labels = x_small.to_numpy()[:,0]

x_dict = {}
for i,j in enumerate(x_numpy):
    x_dict[i] = j[~np.isnan(j)]

sns.set_context("talk")
sns.set_palette("muted")
duration = [i/360 for i in size]
durations_df = pd.DataFrame(zip(x_labels_full, duration),
                            columns=["label","duration"])
facet = sns.displot(x="duration", kind="hist",
                    data=durations_df, height=4.5,
                    stat="count", col="label", bins=12,
                    facet_kws={"sharey":False}, aspect=1.25)
ax = facet.axes.flatten()
ax[0].set_title("Class N")
ax[1].set_title("Class SVEB")
ax[2].set_title("Class VEB")
facet.set_axis_labels(x_var="Trukme", y_var="Kiekis")
facet.fig.suptitle("Elektrokardiogramos segmento trukme (s)",
                   size=20)
facet.fig.subplots_adjust(top=.8)

durations_df.groupby("label").describe()

def placeListGen(df):
    arr = df.to_numpy()
    print(arr.shape)
    maxPlace = []
    for i in range(arr.shape[0]):
        row = arr[:,2:][i]
        seq = row[np.logical_not(np.isnan(row))]
```

```
        seq_size = seq.size
        maxInd = np.where(row == np.nanmax(row))
        maxPlace.append(maxInd[0][0]/seq_size)
    return maxPlace

def toDf(betaList, type):
    df = pd.DataFrame(betaList,
    columns=['Didziausios reiksmes momentas / segmento trukme'])
    return df

def plotBoxPlot(df):
    bt = placeListGen(df)
    bt = toDf(bt,"")
    sns.set_theme(style="whitegrid")
    ax = sns.boxplot(x="Didziausios reiksmes momentas / segmento trukme",
    data=bt)
    ax.set_title(
    "Didziausios reiksmes pasiskirstymas segmente"
    )
    plt.show()

plotBoxPlot(df)

df0 = df[df["label"]==0.0].sample(1000)
df1 = df[df["label"]==1.0].sample(1000)
df2 = df[df["label"]==2.0].sample(1000)

def plotBeta(df):
    arr = df.to_numpy()
    betaList = []
    i = 0
    row = arr[:, 2:][i]
    row = row[np.logical_not(np.isnan(row))]
    seq_size = row.size
    x = np.arange(start=0, stop=seq_size, step=1)
    z = np.polyfit(x, row, 1)
    betaList.append(z[0])
    p = np.poly1d(z)
    plt.plot(row)
    plt.plot(x, p(x), "r--")
    plt.title(f"Trendo ivertinys Beta ≈ {round(z[0],3)}")
    plt.show()

def betaListGen(df):
    arr = df.to_numpy()
    betaList = []
    for i in range(arr.shape[0]):
        row = arr[:,2:][i]
        row = row[np.logical_not(np.isnan(row))]
        seq_size = row.size
        x = np.arange(start=0, stop=seq_size, step=1)
        z = np.polyfit(x, row, 1)
        betaList.append(z[0])
    return betaList

def toDf(betaList, type):
    df = pd.DataFrame(betaList,
                      columns=['Beta'])
    df["Tipas"] = type
    return df

beta0 = betaListGen(df0)
beta0 = toDf(beta0,"N")
beta1 = betaListGen(df1)
```

```
beta1 = toDf(beta1,"SVEB")
beta2 = betaListGen(df2)
beta2 = toDf(beta2,"VEB")

full = beta0.append(beta1, ignore_index=True)
full = full.append(beta2, ignore_index=True)
sns.set_theme(style="whitegrid")
ax = sns.boxplot(x="Tipas",
y="Beta", data=full).set_title(
"Trendo ivertiniu pasiskirstymas"
)
plt.show()

#### Pradinis apdorojimas (filtrai)

from scipy.signal import medfilt
sns.set_context("notebook")

def plot_baseline(i):
    one_series = x_dict[i]
    baseline = medfilt(one_series, 71)
    baseline = medfilt(baseline, 215)
    res = one_series-baseline
    plt.plot(res, color="r")
    plt.plot(one_series, color="b")
    plt.plot(baseline, color="g")

def baseline(x):
    baseline = medfilt(x, 71)
    baseline = medfilt(baseline, 215)
    corrected = x-baseline
    return corrected

x_dict_baseline = {}
for k, v in x_dict.items():
    x_dict_baseline[k] = baseline(v)


import scipy.signal as signal

one_series = x_dict_baseline[6]
my_filter = signal.firwin(13, cutoff=35, fs=360)
one_series_filtered = np.convolve(one_series,
 my_filter, mode='valid')
plt.plot(one_series_filtered, color="r")

def noise(x):
    my_filter = signal.firwin(13, cutoff=35, fs=360)
    return np.convolve(x, my_filter, mode='valid')

x_dict_noise = {}
for k, v in x_dict_baseline.items():
    x_dict_noise[k] = noise(v)

fig, ax = plt.subplots(3, 1, figsize=(8, 10))
ax[0].plot(x_dict[6])
ax[0].set_title("Pradinis signalas")
ax[1].plot(x_dict_baseline[6])
ax[1].set_title("Pradines linijos nukrypimo salinimas")
ax[2].plot(x_dict_noise[6])
ax[2].set_title("Auksto daznio triuksmo salinimas")

import biosppy
```

```
def find_r(x):
    res = biosppy.signals.ecg.christov_segmenter(signal=x,
                                                 sampling_rate=360)
    return res[0]

x_dict_r = {}
for k, v in x_dict_noise.items():
    x_dict_r[k] = find_r(v)


fig, ax = plt.subplots(3, 1, figsize=(8, 12))
ax = ax.flatten()

sns.set_palette("Blues")
loc = np.where(x_labels==0)[0]
for i in loc:
    sns.lineplot(x=np.arange(start =-x_dict_r[i][0],
                 stop=len(x_dict_noise[i])-x_dict_r[i][0]),
                                 y=x_dict_noise[i], alpha=0.1, ax=ax[0])
ax[0].set_title("Class N")

loc = np.where(x_labels==1)[0]
for i in loc:
    sns.lineplot(x=np.arange(start =-x_dict_r[i][0],
                 stop=len(x_dict_noise[i])-x_dict_r[i][0]),
                                 y=x_dict_noise[i], alpha=0.1, ax=ax[1])
ax[1].set_title("Class SVEB")

loc = np.where(x_labels==2)[0]
for i in loc:
    sns.lineplot(x=np.arange(start =-x_dict_r[i][0],
                 stop=len(x_dict_noise[i])-x_dict_r[i][0]),
                                 y=x_dict_noise[i], alpha=0.1, ax=ax[2])
ax[2].set_title("Class VEB")

def findUnusual(df):
    arr = df.to_numpy()
    isskirciuDic = {"ind":[], "type":[],
    "point_low":[], "point_high":[],
    "r_more_than_one":[] ,"r_none":[],
    "r_point":[]}
    maxPlace = []
    for i in range(arr.shape[0]):
        row = arr[:,2:][i]
        seq = row[np.logical_not(np.isnan(row))]
        seq_size = seq.size
        row = baseline(seq)
        row = noise(row)
        maxInd = np.where(row == np.nanmax(row))
        maxPlace.append(maxInd[0][0]/seq_size)
        r = find_r(row)
        if(maxInd[0][0]/seq_size>0.8 or
        maxInd[0][0]/seq_size<0.2 or
        len(r)>1 or len(r)==0):
            isskirciuDic["ind"].append(i)
            isskirciuDic["type"].append(arr[i][0])
            isskirciuDic["r_point"].append(r)
            if(maxInd[0][0]/seq_size>0.8):
                isskirciuDic["point_high"].append(True)
            else:
                isskirciuDic["point_high"].append(False)
            if (maxInd[0][0]/seq_size<0.2):
                isskirciuDic["point_low"].append(True)
            else:
```

```
            isskirciuDic["point_low"].append(False)
        if (len(r)>1):
            isskirciuDic["r_more_than_one"].append(True)
        else:
            isskirciuDic["r_more_than_one"].append(False)
        if (len(r)==0):
            isskirciuDic["r_none"].append(True)
        else:
            isskirciuDic["r_none"].append(False)
return pd.DataFrame(data=isskirciuDic)
```

2 Priedas. Požymių išskyrimo ir klasifikatorių sudarymo programinis kodas

```
### Pozymiu isskyrimas
import pandas as pd
import numpy as np

#### Wavelet
import pywt
import scipy.stats

def wavelet(x):
    features = []
    db2 = pywt.Wavelet('db2')
    coeffs = pywt.wavedec(x, db2, level=4)
    for i in range(0,len(coeffs)):
        features.append(min(coeffs[i]))
        features.append(min(coeffs[i]))
        features.append(max(coeffs[i])/min(coeffs[i]))
        features.append(np.mean(np.abs(coeffs[i])))
        features.append(np.mean(coeffs[i]))
        features.append(np.var(coeffs[i]))
        features.append(scipy.stats.skew(x, 0, True))
    return np.array(features)

def plot_level(x,level):
    db1 = pywt.Wavelet('db2')
    coeff = pywt.wavedec(x, db1, level=4)
    coeff_keep = coeff[level]
    for i,j in enumerate(coeff):
        coeff[i] = np.zeros_like(coeff[i])
    coeff[level] = coeff_keep
    plt.plot(pywt.waverec(coeff,db1),c="red")

#### Hermite basis functions
from numpy.polynomial.hermite import hermfit
def hbf(x):
    coeffs_hbf = np.zeros(15, dtype=float)
    coeffs_HBF_3 = hermfit(range(0,len(x)), x, 3)
    coeffs_HBF_4 = hermfit(range(0,len(x)), x, 4)
    coeffs_HBF_5 = hermfit(range(0,len(x)), x, 5)
    #coeffs_HBF_6 = hermfit(range(0,len(x)), x, 6)
    coeffs_hbf = np.concatenate((coeffs_HBF_3, coeffs_HBF_4, coeffs_HBF_5))
    return coeffs_hbf


#### RR intervals
def rr(x,r):
    x_len = len(x)
    r_len = len(r)
    if r_len > 1:
        left_r = r[0]
        right_r = r[r_len-1]
        rr_back = left_r
        rr_forward = (x_len - right_r)
        return [rr_back,rr_forward]
    elif r_len == 1:
        rr_back = r[0]
        rr_forward = (x_len - r[0])
        return np.array([rr_back,rr_forward])
```

```
    else:
        return np.array([x_len/2,x_len/2])

#### HOS
def hos(x,r):
    hos_b = [None] * 12
    n_intervals = 6
    if len(r) > 0:
        center = r[0]
    else:
        center = int(len(x)/2)
    winR = len(x) - center
    winL = center - 0
    lagR = int(winR / n_intervals*2)
    lagL = int(winL / n_intervals*2)
    for i in range(0,int(n_intervals/2)):
        if i == int(n_intervals/2)-1:
            interval = x[center+i*lagR:]
            hos_b[i] = scipy.stats.skew(interval, 0, True)
            hos_b[3+i] = scipy.stats.kurtosis(interval, 0, False, True)
            continue
        interval = x[center+i*lagR:center+(i+1)*lagR]
        hos_b[i] = scipy.stats.skew(interval, 0, True)
        hos_b[3+i] = scipy.stats.kurtosis(interval, 0, False, True)
    for i in range(0,int(n_intervals/2)):
        if i == int(n_intervals/2)-1:
            interval = x[0:center-(i*lagL)]
            hos_b[6+i] = scipy.stats.skew(interval, 0, True)
            hos_b[9+i] = scipy.stats.kurtosis(interval, 0, False, True)
            continue
        interval = x[center-((i+1)*lagL):center-(i*lagL)]
        hos_b[6+i] = scipy.stats.skew(interval, 0, True)
        hos_b[9+i] = scipy.stats.kurtosis(interval, 0, False, True)
    return np.array(hos_b)

#### Morph and Stat
def morph_and_stat(x,r):
    features = []
    # morphological
    features.append(min(x))
    features.append(max(x))
    features.append(max(x)/min(x))
    area_positive = np.trapz(x[x>0])
    area_negative = abs(np.trapz(x[x<0]))
    features.extend([area_positive,area_negative])
    mean = np.mean(x)
    median = np.median(x)
    var = np.var(x)
    skew = scipy.stats.skew(x, 0, True)
    kurtosis = scipy.stats.kurtosis(x, 0, False, True)
    counts, _ = np.histogram(x, 20, density=True)
    hist_var = np.var(counts)
    features.extend([mean,median,var,skew,kurtosis,hist_var])
    x_len = len(x)
    r_len = len(r)
    if r_len > 1:
        left_r = r[0]
        right_r = r[r_len-1]
```

```
        rr_back = left_r
        rr_forward = (x_len - right_r)
        features.extend([rr_back,rr_forward])
    elif r_len == 1:
        rr_back = r[0]
        rr_forward = (x_len - r[0])
        features.extend([rr_back,rr_forward])
    else:
        features.extend(([x_len/2,x_len/2]))
    return np.array(features)

#### Interpolated series
def interpolate(x,r_points):
    if len(r_points) < 1:
        r_points = np.array([int(len(x / 2))])
    r = np.arange(start =-r_points[0],
                                stop=len(x)-r_points[0])
    f = scipy.interpolate.interp1d(r,x,"previous",fill_value="extrapolate")
    interpolated = f(np.arange(-75,75))
    return interpolated

### Klasifikavimas
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.metrics import accuracy_score
from features import *
import biosppy
x = np.load("saved_features/filtered_train.npy")
def get_feature_size_indices(x,use_features = [False,False,False,False]):
    x = x[~np.isnan(x)]
    r_points = biosppy.signals.ecg.christov_segmenter(signal=x,
    sampling_rate=360)[0]
    indices_start = [0]
    indices_start.append(indices_start[-1]+(wavelet(x).size)-1)
    indices_start.append(indices_start[-1]+hbf(x).size)
    indices_start.append(indices_start[-1]+hos(x,r_points).size)
    indices_start.append(indices_start[-1]+morph_and_stat(x,r_points).size)
    indices_list = []
    if use_features[0]:
        indices_list.extend(list(range(indices_start[0],
        indices_start[1]+1)))
    if use_features[1]:
        indices_list.extend(list(range(indices_start[1]+1,
        indices_start[2]+1)))
    if use_features[2]:
        indices_list.extend(list(range(indices_start[2]+1,
        indices_start[3]+1)))
    if use_features[3]:
        indices_list.extend(list(range(indices_start[3]+1,
        indices_start[4]+1)))

    return indices_list

#### Modelio vertinimas

def model_evaluation(labels,predictions,output=True,print_conf_mat=False):
    n_classes = 3
```

```
    conf_mat = metrics.confusion_matrix(labels, predictions)
    acc = accuracy_score(labels, predictions)
    sensitivity = []
    precision = []
    fpr = []
    for i in range(0, n_classes):
        TP = conf_mat[i,i]
        FP = sum(conf_mat[:,i]) - conf_mat[i,i]
        TN = sum(sum(conf_mat)) - sum(conf_mat[i,:])
            - sum(conf_mat[:,i])
            + conf_mat[i,i]
        FN = sum(conf_mat[i,:]) - conf_mat[i,i]
        if TP + FN != 0:
            sensitivity.append(TP / (TP + FN))
        else:
            sensitivity.append(0)
        if TP + FP != 0:
            precision.append(TP / (TP + FP))
        else:
            precision.append(0)
        if TN + FP != 0:
            fpr.append(FP / (TN + FP))
        else:
            fpr.append(0)
    if output:
        print("Accuracy: ",np.round(acc,3))
        print("Sensitivity: ",np.round(sensitivity,3))
        print("Precision: ",np.round(precision,3))
        print("False Positive Rate: ",np.round(fpr,3))
    if print_conf_mat:
        print(conf_mat)
    return (acc, sensitivity, precision, fpr)

#### Logistine regresija

def eval_logistic_model(logistic_model, scaled_features_eval,
                        labels_eval, output=True):
    predict = logistic_model.predict(scaled_features_eval)
    model_evaluation(labels_eval, predict, output)

from sklearn.linear_model import LogisticRegression

def pipeline_logistic(scaled_features_train, labels_train,
                      scaled_features_eval, labels_eval, penalty="none"):
    print("Penalty = " + penalty)
    logistic_model = LogisticRegression(penalty,
    class_weight = "balanced",
    random_state=123)
    logistic_model.fit(scaled_features_train, labels_train)
    eval_logistic_model(logistic_model, scaled_features_eval, labels_eval)
    return logistic_model

import pickle

def feature_subset_logistic_model(
                scaled_features_train, labels_train,
                scaled_features_eval, labels_eval,
                interpolated_train = None, interpolated_eval = None,
                use_features=[False, False, False, False],
```

```
                preprocessing = True, **kwargs):
    feature_names = ["wavelet","hbf","hos","morph_and_stat"]
    ind = get_feature_size_indices(x[0],use_features)
    scaled_features_train = scaled_features_train[:,ind]
    scaled_features_eval = scaled_features_eval[:,ind]
    name = ""
    if interpolated_train is not None:
        scaled_features_train=
        np.column_stack((scaled_features_train,interpolated_train))
        scaled_features_eval=
        np.column_stack((scaled_features_eval,interpolated_eval))
        name = "interpolated"
    name =  name + "_" + str(sum(use_features))
    for i,j in enumerate(use_features):
        if j:
            name = name + "_" + feature_names[i]
    if not preprocessing:
        name = name + "_no_preprocessing"
    print(name)
    logistic_model = pipeline_logistic(scaled_features_train,
                            labels_train,scaled_features_eval,
                            labels_eval,**kwargs)
    print("Number of features used: ", logistic_model.n_features_in_)
    print("\n\n")
    filename = 'saved_models/logistic_model' + name + '.sav'
    pickle.dump(logistic_model, open(filename, 'wb'))
    return logistic_model

from itertools import permutations

def permute_features_logistic(permute,**kwargs):
    perm_iter = permutations(permute)
    feature_subsets = []
    for i in perm_iter:
        if list(i) not in feature_subsets:
            feature_subsets.append(list(i))
    for i in range(0,len(feature_subsets)):
        feature_subset_logistic_model(
                use_features=feature_subsets[i],**kwargs)

#### SVM
from sklearn.svm import SVC

def train_svm(scaled_features_train,labels_train,C_value,gamma_value):
    svm_model = SVC(C=C_value, kernel='rbf', gamma=gamma_value,
                    class_weight='balanced',
                    decision_function_shape="ovo",
                    random_state=123)
    svm_model.fit(scaled_features_train, labels_train)
    return svm_model

def ovo_class_combinations(n_classes):
    class_pos = []
    class_neg = []
    for c1 in range(n_classes-1):
        for c2 in range(c1+1,n_classes):
            class_pos.append(c1)
```

```
            class_neg.append(c2)
    return class_pos, class_neg

def ovo_voting(decision_ovo, n_classes,scheme="voting"):
    predictions = np.zeros(len(decision_ovo))
    class_pos, class_neg = ovo_class_combinations(n_classes)
    counter = np.zeros([len(decision_ovo), n_classes])
    if scheme == "voting":
        for p in range(len(decision_ovo)):
            for i in range(len(decision_ovo[p])):
                if decision_ovo[p,i] > 0:
                    counter[p, class_pos[i]] += 1
                else:
                    counter[p, class_neg[i]] += 1
            predictions[p] = np.argmax(counter[p])
    if scheme == "voting_both":
        for p in range(len(decision_ovo)):
            for i in range(len(decision_ovo[p])):
                counter[p, class_pos[i]] += decision_ovo[p,i]
                counter[p, class_neg[i]] -= decision_ovo[p,i]
            predictions[p] = np.argmax(counter[p])
    return predictions, counter


def eval_svm_model(svm_model,scaled_features_eval,
        labels_eval,scheme="voting",output=True,
        print_conf_mat=False):
    decision_ovo  = svm_model.decision_function(scaled_features_eval)
    predict_ovo, counter  = ovo_voting(decision_ovo, 3, scheme)
    model_evaluation(labels_eval,predict_ovo,output,print_conf_mat)

from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
from sklearn.metrics import f1_score, make_scorer

scoring = make_scorer(f1_score, average="micro")

def cross_validation_svm(scaled_features_train,labels_train):
    n_features_train = 1/scaled_features_train.shape[1]
    params_search = {
    "C": np.logspace(-3,3,7,base=3),
    "gamma": np.append(np.logspace(-4,-1,4),n_features_train)
    }
    results = []
    svm_model = SVC(kernel='rbf',
                    class_weight='balanced',
                    decision_function_shape="ovo", random_state=123)
    grid_search = HalvingGridSearchCV(svm_model,params_search,
    factor=3,scoring=scoring)
    grid_search.fit(scaled_features_train,labels_train)

    return grid_search.best_estimator_

def pipeline_svm(scaled_features_train,labels_train,
                 scaled_features_eval,labels_eval,
                 scheme="voting",c=None,gamma=None):
    if c is None or gamma is None:
        svm_model = cross_validation_svm(scaled_features_train,
        labels_train)
```

```
        print(svm_model.get_params())
    else:
        svm_model = train_svm(scaled_features_train,
        labels_train,c,gamma)
    eval_svm_model(svm_model,scaled_features_eval,
                   labels_eval,scheme,output=True)
    return svm_model

def feature_subset_svm_model(scaled_features_train,
                         labels_train,
                         scaled_features_eval,
                         labels_eval,
                         scheme="voting",
                         c=None, gamma=None,
                         interpolated_train = None,
                         interpolated_eval = None,
                         use_features=[False,False,False,False],
                         preprocessing=True):

    feature_names = ["wavelet","hbf","hos","morph_and_stat"]
    ind = get_feature_size_indices(x[0],use_features)
    scaled_features_train = scaled_features_train[:,ind]
    scaled_features_eval = scaled_features_eval[:,ind]
    name = ""
    if interpolated_train is not None:
        scaled_features_train =
        np.column_stack((scaled_features_train,
        interpolated_train))
        scaled_features_eval =
        np.column_stack((scaled_features_eval,
        interpolated_eval))
        name = "_interpolated"
    name =  name + "_" + str(sum(use_features))
    for i,j in enumerate(use_features):
        if j:
            name = name + "_" + feature_names[i]
    if not preprocessing:
        name = name + "_no_preprocessing"
    print(name)
    svm_model = pipeline_svm(scaled_features_train,
                labels_train,scaled_features_eval,
                labels_eval,scheme,c,gamma)

    print("Number of features used: ",svm_model.n_features_in_)
    print("\n\n")

    filename = 'saved_models/svm_model' + name + '.sav'
    pickle.dump(svm_model, open(filename, 'wb'))
    return svm_model

def permute_features_svm(permute,c=None,
gamma=None,**kwargs):
    perm_iter = permutations(permute)
    feature_subsets = []
    for i in perm_iter:
        if list(i) not in feature_subsets:
            feature_subsets.append(list(i))
    for i in range(0,len(feature_subsets)):
        feature_subset_svm_model(use_features=feature_subsets[i],
```

```
        c=c,gamma=gamma,**kwargs)

from os import listdir
model_names = pd.Series(listdir("saved_models"))

def load_and_evaluate(n_features,scheme="voting_both",
                      print_conf_mat=True,preprocessing=True):
    if preprocessing:
        names_to_use = (model_names.str.contains(
        "svm_model_" + str(n_features))) &
        (~model_names.str.contains("preprocessing"))
        scaled_features_eval = np.load(
        "saved_features/scaled_features_eval.npy")
    else:
        names_to_use = (model_names.str.contains(
        "svm_model_" + str(n_features))) &
        (model_names.str.contains("preprocessing"))
        scaled_features_eval = np.load(
        "saved_features/scaled_features_evalno_preprocessing.npy")
    labels_eval = np.load("saved_features/labels_eval.npy")
    for i in model_names[names_to_use].values:
        features = np.array([False,False,False,False])
        if "wavelet" in i:
            features[0] = True
        if "hbf" in i:
            features[1] = True
        if "hos" in i:
            features[2] = True
        if "morph_and_stat" in i:
            features[3] = True
        print(i)
        svm_model = pickle.load(open("saved_models/" + i, 'rb'))
        print("C=",svm_model.C)
        ind = get_feature_size_indices(x[0],features)
        eval_svm_model(svm_model,scaled_features_eval[:,ind],
        labels_eval,scheme,print_conf_mat=print_conf_mat)
        print("\n")
```

3 Priedas. Modelio eksperimentinio vertinimo programinis kodas

```
### Apdoroto signalo issaugojimas
def get_filtered_for_single_data(x,use_median=True,use_fir=True):
    x_original = x.copy()
    x = x[~np.isnan(x)]
    if use_median:
        baseline = medfilt(x, 71)
        baseline = medfilt(baseline, 215)
        x = x-baseline
    if use_fir:
        fir_filter = signal.firwin(13,cutoff=35,fs=360)
        x = np.convolve(x, fir_filter, mode='valid')
    x_original[:len(x)] = x
    x_original[len(x):] = np.nan
    return x_original
def get_filtered(df,**kwargs):
    df_numpy = df.iloc[:,2:].to_numpy()
    full_filtered = get_filtered_for_single_data(df_numpy[0],**kwargs)
    full_filtered = np.zeros((len(df),full_filtered.size))
    for i in range(len(df_numpy)):
        full_filtered[i] =
        get_filtered_for_single_data(df_numpy[i],**kwargs)
    return full_filtered
def save_filtered(train_df,test_df,**kwargs):
    results = get_filtered(train_df,**kwargs)
    filtered_train = results
    results = get_filtered(test_df,**kwargs)
    filtered_eval = results
    name = ""
    for k in kwargs.keys():
        name = name + k
    np.save("saved_features/filtered_train" + name +
    ".npy",filtered_train)
    np.save("saved_features/filtered_eval" + name +
    ".npy",filtered_eval)
    return (filtered_train, filtered_eval)
x, x_eval = save_filtered(x,x_eval)
x_no_preprocessing, x_eval_no_preprocessing=
save_filtered(x,x_eval,use_median=False,use_fir=False)
np.save("saved_features/labels_train.npy",x.iloc[:,0].values)
np.save("saved_features/labels_eval.npy",x_eval.iloc[:,0].values)
def get_features_for_single_data(x):
    x = x[~np.isnan(x)]
    try:
        r_points = biosppy.signals.ecg.christov_segmenter(
        signal=x,
        sampling_rate=360)[0]
    except:
        r_points = [int(len(x)/2)]
    features = np.array([])
    features = np.concatenate((features,wavelet(x)))
    features = np.concatenate((features,hbf(x)))
    features = np.concatenate((features,hos(x,r_points)))
    features = np.concatenate((features,morph_and_stat(x,r_points)))
    return features
def get_features(x):
    full_features = get_features_for_single_data(x[0])
```

```
    full_features = np.zeros((len(x),full_features.size))
    for i in range(len(x)):
        full_features[i] = get_features_for_single_data(x[i])
    return full_features
from sklearn.preprocessing import StandardScaler
def save_computed_features(train_x,eval_x,title_addition = ""):
    results = get_features(train_x)
    features_train =  results
    results = get_features(eval_x)
    features_eval = results
    scaler = StandardScaler()
    scaler.fit(features_train)
    scaled_features_train = scaler.transform(features_train)
    scaled_features_train = np.nan_to_num(scaled_features_train)
    scaled_features_eval = scaler.transform(features_eval)
    scaled_features_eval = np.nan_to_num(scaled_features_eval)
    np.save("saved_features/scaled_features_train" +
    title_addition +".npy",scaled_features_train)
    np.save("saved_features/scaled_features_eval" +
    title_addition + ".npy",scaled_features_eval)
    return (scaled_features_train, scaled_features_eval)
scaled_features_train, scaled_features_eval =
                save_computed_features(x,x_eval)
scaled_features_train_no_preprocessing,
scaled_features_eval_no_preprocessing =
            save_computed_features(x_no_preprocessing,
            x_eval_no_preprocessing,
            "no_preprocessing")
def get_interpolated_for_single_data(x):
    x = x[~np.isnan(x)]
    r_points = biosppy.signals.ecg.christov_segmenter(signal=x,
    sampling_rate=360)[0]
    interpolated = interpolate(x,r_points)
    return interpolated
def get_interpolated(x):
    full_interpolated = get_interpolated_for_single_data(x[0])
    full_interpolated = np.zeros((len(x),full_interpolated.size))
    for i in range(len(x)):
        full_interpolated[i] = get_interpolated_for_single_data(x[i])
    return full_interpolated
def save_interpolated(train_df,eval_df,title_addition=""):
    results = get_interpolated(train_df)
    interpolated_train = results
    results = get_interpolated(eval_df)
    interpolated_eval = results
    np.save("saved_features/interpolated_train" +
    title_addition +
    ".npy",interpolated_train)
    np.save("saved_features/interpolated_eval" +
    title_addition +
    ".npy",interpolated_eval)
    return (interpolated_eval,interpolated_train)
interpolated_train, interpolated_eval = save_interpolated(x,x_eval)
interpolated_train_no_preprocessing, interpolated_eval_no_preprocessing =
            save_interpolated(x_no_preprocessing,
                              x_eval_no_preprocessing,
                              "no_preprocessing")
x = np.load("saved_features/filtered_train.npy")
```

```
x_eval = np.load("saved_features/filtered_eval.npy")
scaled_features_train = np.load("saved_features/scaled_features_train.npy")
labels_train = np.load("saved_features/labels_train.npy")
scaled_features_eval = np.load("saved_features/scaled_features_eval.npy")
labels_eval = np.load("saved_features/labels_eval.npy")
interpolated_train = np.load("saved_features/interpolated_train.npy")
interpolated_eval = np.load("saved_features/interpolated_eval.npy")
x_no_preprocessing = np.load("saved_features/" +
                        "filtered_trainuse_medianuse_fir.npy")
x_eval_no_preprocessing = np.load("saved_features/" +
                         "filtered_evaluse_medianuse_fir.npy")
scaled_features_train_no_preprocessing=
np.load("saved_features/scaled_features_trainno_preprocessing.npy")
scaled_features_eval_no_preprocessing=
np.load("saved_features/scaled_features_evalno_preprocessing.npy")
interpolated_train_no_preprocessing=
np.load("saved_features/interpolated_trainno_preprocessing.npy")
interpolated_eval_no_preprocessing=
np.load("saved_features/interpolated_evalno_preprocessing.npy")
outliers_train = np.load("saved_features/outliersTrain.pkl",
allow_pickle=True)
outliers_test = np.load("saved_features/outliersTest.pkl",
allow_pickle=True)
### Pozymiu vizualizavimas
from matplotlib.ticker import MaxNLocator
def plot_feature_mean(feature,labels,which,ax=None):
    y = x[0]
    feature_subsets = [[True,False,False,False],[False,True,False,False],
    [False,False,True,False],[False,False,False,True]]
    indices = get_feature_size_indices(y,feature_subsets[which])
    groups = np.array(["Bangeliu transformacija",
                       "Ermito bases koeficientai",
                       "Aukstesnes eiles statistikos",
                       "Morfologiniai ir statistiniai pozymiai"])
    df = pd.DataFrame(feature).iloc[:,indices]
    df.columns = range(df.columns.size)
    df["ėKlas"] = pd.Series(labels).replace({0:"N",1:"SVEB",2:"VEB"})
    df2 = df.groupby("Klase").mean().reset_index().melt("Klase")
    if ax is None:
        fig, ax = plt.subplots(1,1,figsize=(7, 5))
    ax = sns.lineplot(x="variable",y="value",
    hue="Klase",data=df2,palette="Set2",ax=ax)
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.set_title(groups[which])
plot_feature_mean(scaled_features_train,labels_train,0)
plot_feature_mean(scaled_features_train,labels_train,1)
plot_feature_mean(scaled_features_train,labels_train,2)
plot_feature_mean(scaled_features_train,labels_train,3)
fig, ax = plt.subplots(2,2,figsize=(16, 10))
ax = ax.flatten()
for i in range(0,4):
    plot_feature_mean(scaled_features_train,labels_train,i,ax=ax[i])
### Klasifikatoriu ivertinimas
#### Logistine regresija
# vien tik signalas be pozymiu (baseline modelis)
permute_features_logistic([False,False,False,False],
```

```
                            scaled_features_train=scaled_features_train,
                            interpolated_train = interpolated_train,
                            labels_train = labels_train,
                            scaled_features_eval = scaled_features_eval,
                            interpolated_eval = interpolated_eval,
                            labels_eval = labels_eval)
# Naudojant po viena pozymiu grupe
permute_features_logistic([True,False,False,False],
                            scaled_features_train=scaled_features_train,
                            labels_train = labels_train,
                            scaled_features_eval = scaled_features_eval,
                            labels_eval = labels_eval)
# Naudojant po dvi pozymiu grupes
permute_features_logistic([True,True,False,False],
                            scaled_features_train=scaled_features_train,
                            labels_train = labels_train,
                            scaled_features_eval = scaled_features_eval,
                            labels_eval = labels_eval)
# Naudojant po tris pozymiu grupes
permute_features_logistic([True,True,True,False],
                            scaled_features_train=scaled_features_train,
                            labels_train = labels_train,
                            scaled_features_eval = scaled_features_eval,
                            labels_eval = labels_eval)
# Naudojant visas pozymiu grupes
permute_features_logistic([True,True,True,True],
                            scaled_features_train=scaled_features_train,
                            labels_train = labels_train,
                            scaled_features_eval = scaled_features_eval,
                            labels_eval = labels_eval)
# Naudojant vien tik signala be pradinio duomenu tvarkymo
permute_features_logistic([False,False,False,False],
        scaled_features_train=scaled_features_train_no_preprocessing,
        interpolated_train = interpolated_train_no_preprocessing,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval_no_preprocessing,
        interpolated_eval = interpolated_eval_no_preprocessing,
        labels_eval = labels_eval,
        preprocessing = False )
# Naudojant po viena pozymiu grupe be pirminio tvarkymo
permute_features_logistic([True,False,False,False],
         scaled_features_train=scaled_features_train_no_preprocessing,
         labels_train = labels_train,
         scaled_features_eval = scaled_features_eval_no_preprocessing,
         labels_eval = labels_eval,
         preprocessing = False)
# Naudojant visus pozymius be pirminio tvarkymo
permute_features_logistic([True,True,True,True],
         scaled_features_train=scaled_features_train_no_preprocessing,
         labels_train = labels_train,
         scaled_features_eval = scaled_features_eval_no_preprocessing,
         labels_eval = labels_eval,
         preprocessing = False)
#### SVM
permute_features_svm([False,False,False,False],
         scheme="voting_both",c=1,gamma=1/150,
         scaled_features_train=scaled_features_train,
         labels_train = labels_train,
         scaled_features_eval = scaled_features_eval,
         labels_eval = labels_eval,
```

```
        interpolated_train = interpolated_train,
        interpolated_eval = interpolated_eval)
svm_model = pickle.load(open("saved_models/svm_modelinterpolated_0.sav",
                             'rb'))
eval_svm_model(svm_model,interpolated_eval,
           labels_eval,scheme="voting",
           print_conf_mat=True)
permute_features_svm([True,False,False,False],
        scheme="voting_both",c=None,gamma=None,
        scaled_features_train=scaled_features_train,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval,
        labels_eval = labels_eval)
load_and_evaluate(1,scheme="voting")
permute_features_svm([True,True,False,False],
        scheme="voting_both",c=None,gamma=None,
        scaled_features_train=scaled_features_train,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval,
        labels_eval = labels_eval)
load_and_evaluate(2,scheme="voting")
permute_features_svm([True,True,True,False],scheme="voting_both",
        c=None,gamma=None,
        scaled_features_train=scaled_features_train,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval,
        labels_eval = labels_eval)
load_and_evaluate(3,scheme="voting")
permute_features_svm([True,True,True,True],scheme="voting_both",
        c=None, gamma=None,
        scaled_features_train=scaled_features_train,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval,
        labels_eval = labels_eval)
load_and_evaluate(4,scheme="voting")
permute_features_svm([False,False,False,False],
         scheme="voting_both",c=1,gamma=1/150
        scaled_features_train=scaled_features_train_no_preprocessing,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval_no_preprocessing,
        labels_eval = labels_eval,
        interpolated_train = interpolated_train_no_preprocessing,
        interpolated_eval = interpolated_eval_no_preprocessing,
        preprocessing = False)
svm_model = pickle.load(open("saved_models/"+
       "svm_model_interpolated_0_no_preprocessing.sav", 'rb'))
eval_svm_model(svm_model,interpolated_eval,labels_eval,
              scheme="voting",print_conf_mat=True)
permute_features_svm([True,False,False,False],
        scheme="voting_both",c=None,gamma=None,
        scaled_features_train=scaled_features_train_no_preprocessing,
        labels_train = labels_train,
        scaled_features_eval = scaled_features_eval_no_preprocessing,
        labels_eval = labels_eval,
        interpolated_train = interpolated_train,
        interpolated_eval = interpolated_eval,
        preprocessing=False)
load_and_evaluate(1,scheme="voting",preprocessing=False)
permute_features_svm([True,True,True,True],
```

```
            scheme="voting_both",c=None,gamma=None,
            scaled_features_train=scaled_features_train_no_preprocessing,
            labels_train = labels_train,
            scaled_features_eval = scaled_features_eval_no_preprocessing,
            labels_eval = labels_eval,
            preprocessing=False)
load_and_evaluate(4,scheme="voting",preprocessing=False)
```