

VILNIAUS UNIVERSITETAS  
MATEMATIKOS IR INFORMATIKOS FAKULTETAS  
PROGRAMŲ SISTEMŲ KATEDRA

# Neuroninių tinklų panaudojimas testavimo proceso gerinimui

# Using neural networks to improve the testing process

## Bakalauro darbas

Atliko: Ričardas Mikelsonis (parašas)  
Darbo vadovas: asist. dr. Vytautas Valaitis (parašas)  
Darbo recenzentas: partn. prof., dr. Aldas Glemža (parašas)

Vilnius – 2018

## **Santrauka**

Glaustai aprašomas darbo turinys: pristatoma nagrinėta problema ir padarytos išvados. Santraukos apimtis ne didesnė nei 0,5 puslapio. Santraukų gale nurodomi darbo raktiniai žodžiai.

**Raktiniai žodžiai:** **raktinis žodis 1, raktinis žodis 2, raktinis žodis 3, raktinis žodis 4, raktinis žodis 5**

## **Summary**

Santrauka anglų kalba. Santraukos apimtis ne didesnė nei 0,5 puslapio.

**Keywords:** keyword 1, keyword 2, keyword 3, keyword 4, keyword 5

## TURINYS

ĮVADAS .....	4
1. PROGRAMINIO KODO SUDĒTINGUMO IR DEFEKTYVUMO SĄSAJOS .....	5
1.1. McCabe sudėtingumo metrikos .....	6
1.2. Halstead sudėtingumo metrikos .....	7
2. SUDĒTINGUMO METRIKŲ PANAUDOJIMAS APRAŠANT PROGRAMINĖS ĮRANGOS DEFEKTYVUMĄ NUSPĖJANČIĄ BESIMOKANČIĄ MAŠINĄ .....	8
2.1. Sprendimų medžio metodu paremta besimokanti mašina .....	9
2.2. Daugiasluoksnis perceptronas .....	10
3. DUOMENŲ ATRINKIMAS .....	11
3.1. Rankiniu būdu ieškant atributų kurie yra tiesiogiai proporcingi kitiems atributams ....	11
3.2. Duomenų analizė naudojant įrankį Weka .....	11
4. NEURONINIO TINKLO EFEKTYVUMO GERINIMAS NAUDOJANT ATRINKTUS DUOMENIS .....	12
REZULTATAI IR IŠVADOS .....	13
SANTRUMPOS .....	14
PRIEDAI .....	14
1 priedas. Sprendimų medžio besimokanti mašina .....	15
2 priedas. Eksperimentinio palyginimo rezultatai .....	16

## Ivadas

Smarkiai augant Continuous Delivery principų popularumui auga poreikis testuoti daugiau per trumpesnį laiko tarą. Paprastas sprendimas šiai problemai būtų be abejo nuolat besiskantis automatinių testų paketas, tačiau automatinius testus padengti programinį kodą 100% praktiškai neįmanoma. 100% padengimas įmanomas tik pagal kokią nors specifinę metriką, o vaikytis tik-rojo 100% padengtumo testais pagal kiekvieną metriką ar funkcinę sritį prilygsta šviesos geičio vaikymuisi, kuo arčiau esame tikslu tuo daugiau pastangų ir resursų reikia pasistumėti į priekį. Todėl, žinoma, vis dar išlieka poreikis rankiniams testavimams. Norint išleisti programinės įrangos versiją kuo dažniau ištenuoti visko kiekvieną kartą neįmanoma. Taip šis problemos sprendimas iškelia dar vieną, mažesnę, problemą: kaip efektyviai pasirinkti testavimo sritis kiekvienai naujai programos laidai (angl. release). Alan M. Davis [**Davis:1995:PSD:203406**] tegia, jog pareto principą galima pritaikyti ir programinės įrangos testavime: čia 80% kode esančių defektų surandami 20% viso kodo.

Tobulame pasaulyje iteracijai pasirinktos testavimo sritys ir apims tuos 20% problematiškojo kodo. Tačiau dažnai net remiantis visa turima istorine informacija testuotojui atsakingam už testavimo sričių parinkimą yra sunku efektyviai atrinkti testavimo sritis, kuriose atliktas darbas turės didžiausią įtaką programinės įrangos kokybei.

—

Siekiant sėkmingai įgyvendinti darbo tikslą bus įgyvendinti šie uždaviniai:

1. Išrinkti kodo metrikas darančias įtaką programinės įrangos defektyvumui
2. Palyginti efektyvumą tarp sprendimų medžio besimokančios mašinos ir neuroninio tinklo naudojant surinktus duomenis
3. Atrinkti bei atmesti duomenis turinčius mažiausią įtaką galutiniams spėjimui
4. Naudojantis mažesniu atributų kiekiu iš naujo apmokyti neuroninį tinklą ir palyginti rezultatus su pilnų duomenų rinkinių apmokyto neuroninio tinklo

# 1. Programinio kodo sudėtingumo ir defektyvumo sąsajos

Kiekvieną kartą renkatis testuojamas programinės įrangos vietas, žinoma, jei nėra testuojamas visas programinės įrangos funkcionalumas, reikia turėti atskaitos tašką, kuris leistų nuspresti kurios programinės įrangos kodo vėtos, ar programų paketo moduliai turi didžiausią riziką būti defektyvūs. I šią kategoriją dažnai pakliūva seniai testuoti, nauji, atnaujinti bei anksčiau daug defektų turėję programinės įrangos moduliai. Didelė problema, su kuria vis dažniau susiduriama yra ką daryti, jei nėra laiko ištестuoti visiems šiemis moduliams ir kaip tuomet pasirinkti atitinkamus programinės įrangos modulius testavimui, kad nešvaistydami laiko padarytumėme didžiausią įmanomą įtaką programinės įrangos kokybės užtikrinimui.

Visų pirma reikėtų sukurti detalesnę programinės įrangos metrikų sistemą nei, „naujumas“ ar „prieš tai buvusių defektų kiekis“, kuri tiesiogiai atspindėtų būsimą (ar esamą) programinės įrangos defektyvumą. O gal būt užtenka pritaikyti egzistuojančias programinio kodo metrikas randant koreliaciją su programų defektyvumu. Ir iš tiesų, būtų galima teigti, jog programinės kokybės metrikos kurias būtų įmanojma tiesiogiai susieti su programinio kodo defektyvumu egzituoja. Arčiausiai tokį metrikų yra programino kodo sudėtingumo metrikos. Be abejo yra ne vienas būdas matuoti programos kodo sudėtingumą, tačiau, šiam tyrimui pasirinktos dvi, galima būtų sakyti, populiarusios metrikos kodo sudėtingumui atvaizduotii: Maurice H. Halstead aprašytosios 1977-aisiais [**Halstead:1977:ESS:540137**] bei Thomas J. McCabe aprašytosios 1976-aisiais [**McCabe:1976:CM:800253.807712**].

Ryšys tarp kodo sudėtingumo bei defektų buvimo tame, o gal būt derėtų sakyti defektų buvimo galimybės kode, atrodytų, gana logiškas: kuo sudėtingesnis kodas, tuo sunkiau jį skaityti, kuo sunkiau kodas skaitomas tuo sunkiau jį plėsti, todėl kyla rizika defektų atsiradimui. Tokia prielaida tiriama nuo pat sudėtingumo metrikų aprašymo. Žinoma, dėl kodo sudėtingumo tiksliai defektų kiekui daromo efekto kyla nestuarimų. Atsiranda teigiančių, jog Halstead metrikos neturi jokio įrodomo ryšio su defektų buvimu, tačiau yra ir tyrimų pagrindžiančių kodo sudėtingumo bei defektų buvimo tame koreliaciją [**Schroeder1999APG**]. Dažniausiai tyrėjai lieka šio argumento viduryje nei visiškai neigdami, nei patvirtindami šią koreliaciją. Gana dažnas sprendimas yra pamažinti metrikų kiekį ir patikrinti metrikas priklausomai nuo situacijos, taip užtikrinant, kad pasirinktosios metrikos tikrai korelioja su tyrimo metu ieškomu fenomenu, šiuo atveju kodo defektyvumu [**Metrics in Evaluating Software Defects:2013**].

Tokių metrikų pasirinkimas paremtas tuo, jog jų rinkimas, net atliekamas nuolat nereikalauja daug papildomo laiko, išteklių ar pastangų. Tai statinės kodo metrikos, kurias gana paprasta surinkti su kodo analizės įrankiais ir kurios salyginai tikliai parodo kodo defektyvumą.

Šiame dokumente aprašomame tyime bus naudojamos visos Halstead ir McCabe metrikos kartu, bei keletas šių atributų poaibį susiaurintų tiek rankiniu būdų tiek duomenų analizės įrankiais.

## 1.1. McCabe sudėtingumo metrikos

1976-aisiais Thomas J, McCabe išleido tyrimą kuriame pasiūlė naują programinės įrangos kodo sudėtingumo matavimo būdą, pavadintą Ciklomatiniu sudėtingumu (angl. Cyclomatic Complexity). Šis matavimo vienetas kiekybiškai matuoja nepriklausomų kelių per kodo elementus kiekį.

Ciklomatnis sudėtingumas apibrėžiamas, kaip kodo vykdyme egzistuojančių tiesiškai nepriklausomų kelių kiekis. Pavyzdžiui jei kode nėra jokių eiliškumo kontrolės sakinių (angl. Control flow statements), kaip sąlygniniai „if“ sakiniai, tuomet kodo ciklomatinis sudėtingumas būtų 1. Su vienu „if“ sąlyginiu sakiniu, po kurio įvykdymo galimi du keliai: jei grąžinta „TRUE“ bei jei grąžinta „FALSE“, kodo ciklomatinis sudėtingumas būtų 2. Du vieną sąlygą turintys „if“ sakiniai ar vienas toks sakinsky su dviem salygomis sudarytų grafą kurio ciklomatinis sudėtingumas būtų lygus 3 ir t.t. [McCabe:1976:CM:800253.807712]

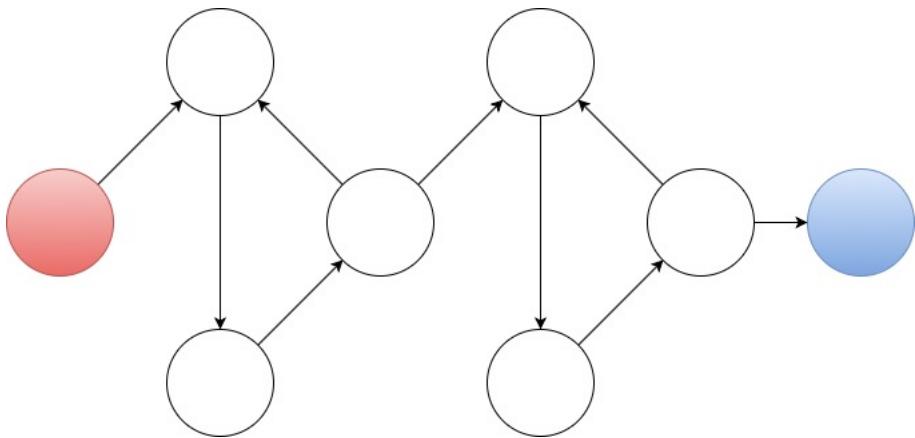
Ciklomatinis sudėtingumas skaičiuojamas naudojant orientuotą grafą, kuris vaizduoja programos kodą. Tokiu atveju grafo viršūnės atspindi komandų, kurios vykdomos kartu, grupes, o kraštinię su kryptimi sujungia dvi viršūnes jei jos gali būti vykdomos viena po kitos. Toki ciklomatinio sudėtingumo skaičiavimo metodą galima taikyti ir aukštesnės abstrakcijos progaminio kodo vienetams kaip funkcijos, metodai, klasės ar kt.

Matematiškai ciklomatinis sudėtingumas M apskaičiuojamas formule  $M = E - N + 2P$ .  
Kur šios metrikos atvaizduojamos taip:

- .  $E$  = grafo kraštinių kiekis,
- .  $N$  = grafo viršunių kiekis,
- .  $P$  = apjungtų komponentų kiekis.

Dar ciklomatinis sudėtingumas skaičiuojamas ir formule  $M = E - N + P$ . Šiuo atveju kodas vaizduojamas grafu kurio pradinio ir paskutinio modulio grafo viršūnės yra sujungtos, taip sudarant stipriai apjungtą grafą, ir kodo ciklomatinis sudėtingumas prilygsta grafo ciklomatiniam skaičiui.

Pavyzdžiui, turint programą, kaip pavaizduota pav. 1 kurios veikimas prasideda nuo modulio atvaizduoto raudona viršūne, o kodo vykdymas pabaigiamas įvykdžius mėlyna viršūne atvaizduotą kodą galime panaudoti McCabe formulę ir apskaičiuoti kodo ciklomatinį sudėtingumą. Toks grafas sudarytas iš devynių kraštinių bei aštuonių viršunių, kurie kartu sudaro vieną apjungtajį komponentą, tuomet tokios programos ciklomatinis sudėtingumas būtų  $9 - 8 + 2 * 1 = 3$ .



1 pav. Programinio kodo atvaizdavimas grafu

## 1.2. Halstead sudėtingumo metrikos

1977-aisiais Maurice H. Halstead, knygoje „Elements of Software science“ pateikė kodo sudėtingumo metrikas kaip būdą programinės įrangos kūrimą paversti empiriniu mokslu. Halstead metrikos tiesiogiai priklauso nuo to, kaip kodas parašytas ir yra skaičiuojamos statiškai, pagal operatorius bei operandus. Pasak M. H. Halstead kompiuterinė programa yra tiesiog algoritmo realizacija, sudaryta iš operatorių bei operandų sekos.**[Halstead:1977:ESS:540137]**

Halstead metrikos paremtos šiais matavimo vienetais:

- .  $n_1$  - unikalių operatorių kiekis,
- .  $n_2$  - unikalių operandų kiekis,
- .  $N_1$  - visų operatorių kiekis,
- .  $N_2$  - visų operandų kiekis.

Iš šių matų šskaičiuojami šios metrikos:

- .  $n = n_1 + n_2$  - Programos žodynas,
- .  $N = N_1 + N_2$  - programos dydis,
- .  $V = N * \log_2 n$  - programos apimtis,
- .  $D = \frac{n_1}{2} * \frac{N_1}{n_2}$  - programos sudėtingumas,
- .  $E = D * V$  - pastangos.

Čia programos sudėtingumas reiškia sudėtingumą skaitant ar rašant tolesnį kodą pvz. atliekant kodo peržiūrą.

Halstead metrikos dažnai kritikuojamas dėl neapibrėžto operatorių bei operandų nustatymo būdų, dažnai kritikos susilaukia ir tai, jog matuojamas tekstu pagrįstas sudėtingumas, t.y. kaip sunku skaityti kodą, o ne sudėtingumas pagrįstas kodo struktūra ar logika.

## **2. Sudėtingumo metrikų panaudojimas aprašant programinės įrangos defektyvumą nuspėjančią besimokančią mašiną**

Turint aiškiai apibrėžtas metrikas programinės įrangos kodui apibūdinti galima būtų vesti žurnalą kiekvienam programinės įrangos moduliui ir kiekvienu kartą renkantis programinės įrangos sritis, kurias reikia testuoti, palyginti esamos kodo laidos (angl. release) metrikas su žurnale esančiomis taip nuspriandžiant ar išanalizuotą sriti reikia testuoti dabar ar sričiai priskirti žemesnę pirmenybę. Tačiau toks procesas atliekamas rankiniu būdu ne tik, kad ne taupo laiko, tačiau ir prideda papildomų rūpesčių testuotojams sprendžiantiems testų prioritetus. Iš to turime problemą kuri dažnai sprendžiama automatizuojant ir viską aprašant besimokančią mašiną. Šiame tyrime bus naudojamos dviejų tipų besimokančios mašinos: Sprendimų medžio spėjamasis modelis bei daugiasluoksniu perceptronu aprašytą neuroninį tinklą.

Apmokyti šias besimokančiasias mašinas bus naudojama kombinacija atributų gautų kodo statinės analizės metu, ir Halstead bei McCabe metrikų apskaičiuotų naudojant statinės analizės metu surinktus duomenis. Iš viso duomenų lentelė kuri bus pateikta besimokančiai mašinai susidės iš 22 stulpelių: 21 atributo bei 1 stulpelio reiškiančio defektų egzistavimą. Reikšmės žymimos taip:

- . loc - Skaitinė reikšmė. McCabe eilučių kode kiekis.
- . v(g) - Skaitinė reikšmė. McCabe ciklomatinis sudėtingumas.
- . ev(g) - Skaitinė reikšmė. McCabe esminis sudėtingumas.
- . iv(g) - Skaitinė reikšmė. McCabe dizaino sudėtingumas.
- . n - Skaitinė reikšmė. Halstead visų operatoriu bei operandų suma (programos dydis).
- . v - Skaitinė reikšmė. Halstead programos apimtis.
- . l - Skaitinė reikšmė. Halstead programos ilgis.
- . d - Skaitinė reikšmė Halstead sudėtingumas.
- . i - Skaitinė reikšmė. Halstead protingo turinio metrika.
- . e - Skaitinė reikšmė. Halstead pastangos.
- . b - Skaitinė reikšmė. Halstead defektų kiekių spėjimas.
- . t - Skaitinė reikšmė. Halstead laiko suprogramuoti metrika.
- . lOCode - Skaitinė reikšmė. Halstead kodo eilučių kiekis.
- . lOComment - Skaitinė reikšmė. Halstead komentarų eilučių kiekis.
- . lOBlank - Skaitinė reikšmė. Halstead tuščių eilučių kiekis.
- . lOCodeAndComment - Skaitinė reikšmė. Komentarų bei kodo eilučių kiekių sumą.
- . uniq\_Op - Skaitinė reikšmė. Unikalių operatorių kiekis.
- . uniq\_Opnd - Skaitinė reikšmė. Unikalių operandų kiekis.
- . total\_Op - Skaitinė reikšmė. Visų operatorių kiekis.
- . total\_Opnd - Skaitinė reikšmė. Visų operandų kiekis.
- . branchCount - Skaitinė reikšmė. Grafu atvaizduojamo kodo kelių kiekis.
- . defects - Binarinė reikšmė. True arba False.

Tyrimo tikslui pasiekti buvo nuspresta panaudoti keletą jau surinktų duomenų rinkinių. Siekiant surinkti pakankamą kiekį prasmingų duomenų (500-1000 skirtinę testuotų programinės įrangos versijų (angl. build)) šio tyrimo kontekste užimtu per daug laiko. Tyrime bus naudojami „CM1“ bei „PC1“ duomenų rinkiniai iš „Promise“ duomeų rinkinių duomenų bazės [Sayyad-Shirabad+Menzies:2005]. Tieki „CM1“, tiek „PC1“ pateikti NASA ir sudaryti jų 2004-aisiais vykdytos „NASA Metrics Data Program“ progamos, skirtos kurti defektus nuspėjančią programinę įrangą, metu.

PC1 duomenų rinkinys sudarytas iš 1109 duomenų eilučių, surinktų iš kodo parašyto C programavimo kalba. Programinė įranga iš kurios surinkti šie duomenys skirta orbitoje skraidančiam palydovui kontroliuoti. Duomenų rinkinys pasidalina į 6.94% duomenų eilučių kurios atspindi atvejį, kai užfiksotas vienas ar daugiau defektų bei 93.06% duomenų eilučių atspindinčių, kai nebuvo užfiksota defektų.

CM1 duomenų rinkinys sudarytas iš 498 duomenų eilučių, surinktų iš kodo parašyto C programavimo kalba. Programinė įranga iš kurios surinkti šie duomenys yra pagalbinė įranga vienam iš NASA erdvėlaivių. Duomenų rinkinys pasidalina į 90.16% duomenų eilučių kurios atspindi atvejį, kai užfiksotas vienas ar daugiau defektų bei 9.83% duomenų eilučių atspindinčių, kai nebuvo užfiksota defektų.

## 2.1. Sprendimų medžio metodu paremta besimokanti mašina

Pirmasis prediktyvusis modelis paremtas sprendimų medžio klasifikatoriumi. Toks modelis yra laisvai suprantamas ir skaitomas net ir paprastam žmogui. Vienas tokį modelių sugeneruotas naudojant PC1 duomenų rinkinio poaibį pavaizduotas pav. 2 pirmame priede.

Su tradiciniais duomenų rinkiniais sprendimų medis turi nemažai privalumų. Sprendimo medžiai yra lengvai suprantami, net daug techninių žinių neturintiems žmonėms ir pateikus sprendimų medžių grafiškai pačiam galima pereiti per viršunes tikrinant duomenų rinkinio eilutės grąžinamą rezultatą. Taip pat palyginus su kitais prediktyviaisiais modeliais sprendimo medžiai reikalauja salyginai nedidelio kieko duomenų pertvarkymo prieš naudojant juos šiame modelyje. Iki tam tikro laipsnio smarkiai išskiriančios reikšmės ar trūstami įrašai nedaro didelės įtakos sprendimų medžiams. Sprendimų medžiai neskiria ir gali priimti tiek skaitines tiek logines reikšmes, todėl galima naudoti didesnę įvairovę duomenų rinkinių.

Didžiausi sprendimų medžio trūkumai yra tikslumo trūkumas, kadangi tai, ginčytinai, paprasčiausias besimokančios mašinos modelis jis yra ir pats netiksliausias. Modelio paprastumas su dideliais duomenų rinkiniais gali privesti prie per sudėtingo medžio sugeneravimo, kuomet nebeatpažystami ryšiai tarp atributų bei galutinio rezultato. Tai dažnai vadinama permokymu (angl. overfitting).

Iprastai sprendimų medžio prediktyvusis modelis naudojamas salyginai paprastų duomenų rinkinių problemoms spręsti. Žemo lygio klasifikatoriams bei prediktyviesiems modeliams, kaip Iriso gėlės skirtinę rūšių klasifikavimo problema.

Rezultatai apskaičiuojami keletu formuliu:

$$\cdot Precision = \frac{\Sigma T_{ikraspozityvas}}{\Sigma T_{ikraspozityvas} + \Sigma N_{etikraspozityvas}}$$

- .  $Accuracy = \frac{\Sigma T_{ikraspozityvas} + \Sigma T_{ikrasnegatyvas}}{\Sigma T_{ikraspozityvas} + \Sigma T_{ikrasnNegatyvas} + \Sigma N_{etikraspozityvas} + \Sigma N_{etikasnegatyvas}}$
- .  $Recall = \frac{\Sigma T_{ikraspozityvas}}{\Sigma T_{ikraspozityvas} + \Sigma N_{etikrasnegatyvas}}$

kur „Tikrasis pozityvas“ yra teisingai nustatyta pozityvi reikšmė, „Tikrasis negatyvas“ yra teisingai nustatyta negatyvi reikšmė, „Netikras pozityvas“ yra klaidingai nustatyta pozityvi reikšmė, o „Netikras negatyvas“ yra neteisingai nustatyta negatyvi reiškmė.

Besimokančių mašinų kontekste Accuracy reiškia kiek procentaliai iš visų spėtų reikšmių buvo atspėta teisinga reikšmė. Precision parodo kiek iš atspėtų teigiamų reikšmių yra tikrų pozityvų, t.y. teisingai nuspėtų teigiamių reikšmių. O Recall parodo kiek tarp iš viso buvusių teigiamų reikšmių jų buvo atspėta.

1 lentelė. Sprendimų medžio mašinos rezultatai

Duomenų rinkinys	accuracy	precision	recall
PC1	90%	30%	35%
CM1	84%	13.3%	21%

Lentelė 1 atvaizduoja accuracy, precision bei recall reikšmes gautas iš sprendimų medžiu paremtos modelio. Modeliui apmokyti ir testuoti abiem atvejais buvo panaudotas visas duomenų rinkinys su 21 atributu, kur 50% atsitiktinai atrinktų reikšmių buvo panaudotos, kaip mokomieji duomenys, o kiti 50%, kaip testiniai.

Sprendimo medžiu paremtas prediktyvus modelis daug nežada. Su CM1 duomenų rinkiniu turime didelį, bet toli gražu ne patį geriausią accuracy, o visais atvejais tiek precision tiek recall reikšmės nesiekė 50%. Galima būtų kelti hipotezę, jog didžiausią problemą sukelia duomenų rinkiniai turintys 21 atributą ir labai nelygū duomenų pasiskirstymą. Tačiau pirmiausia reikėtų patikrinti ar su tokiais pat, ne modifikuotais duomenimis galima hauči gerensius rezultatus naudojant kitokį klasifikatorių.

## 2.2. Daugiasluoksnis perceptronas

### **3. Duomenų atrinkimas**

- 3.1. Rankiniu būdu ieškant atributų kurie yra tiesiogiai proporcingsi kitiems atributams**
- 3.2. Duomenų analizė naudojant įrankį Weka**

**4. Neuroninio tinklo efektyvumo gerinimas naudojant at-rinktus duomenis**

## **Rezultatai ir išvados**

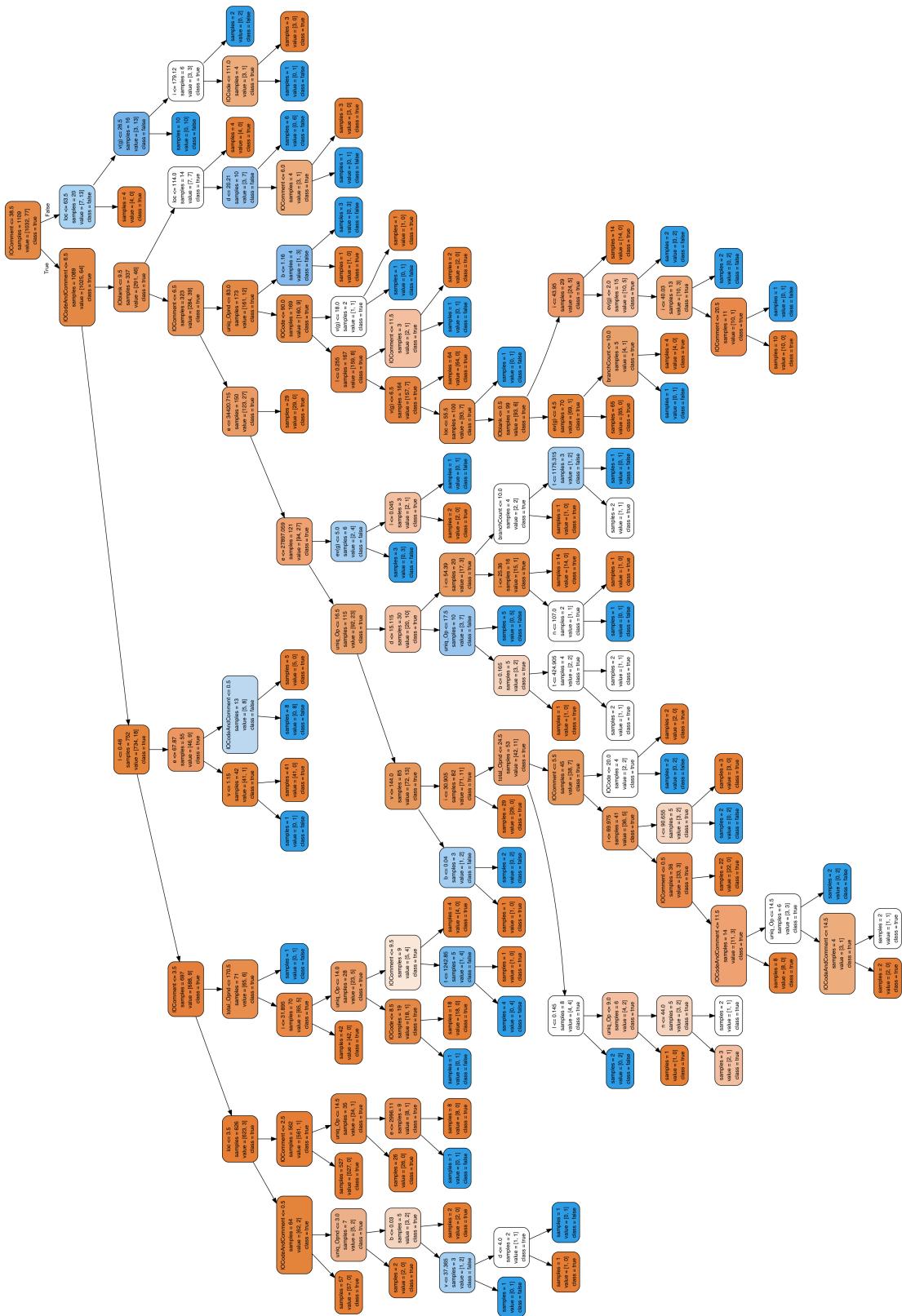
Rezultatų ir išvadų dalyje išdėstomi pagrindiniai darbo rezultatai (kažkas išanalizuota, kažkas sukurta, kažkas įdiegta), toliau pateikiamos išvados (daromi nagrinėtų problemų sprendimo metodų palyginimai, siūlomos rekomendacijos, akcentuojamos naujovės). Rezultatai ir išvados pateikiami sunumeruotų (gali būti hierarchiniai) sąrašų pavidalu. Darbo rezultatai turi atitikti darbo tikslą.

## **Santrumpos**

Savokų apibrėžimai ir santrumpų sąrašas sudaromas tada, kai darbo tekste vartojami specialūs paaiškinimo reikalaujantys terminai ir rečiau sutinkamos santrumpos.

## Priedas nr. 1

## **Sprendimų medžio besimokanti mašina**



2 pav. Sprendimų medžio tipo besimokanti mašina sudaryta naudojant PC1 duomenų rinkinio poaibį

## Priedas nr. 2

### Eksperimentinio palyginimo rezultatai

2 lentelė. Lentelės pavyzdys

Algoritmas	$\bar{x}$	$\sigma^2$
Algoritmas A	1.6335	0.5584
Algoritmas B	1.7395	0.5647