VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Dirbtinių neuroninių tinklų panaudojimas testavimo proceso gerinimui

Using an Artificial Neural Network in the Software Testing Process

Bakalauro darbas

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

Vilnius – 2018

Santrauka

Darbe analizuojamas dirbtinio neuroninio tinklo, apmokyto nuspėti programinės įrangos defektyvumą, kaip įrankio testavimo procese efektyvumas.

Darbas susideda iš keturių esminių dalių. Pirmoje apžvelgiamos kodo matavimo metrikos bei jų ryšys su programinio kodo defektuvumu. Toliau vykdomas eksperimentas bei palyginami rezultatai apmokant bei testuojant sprendimų medžio modelio besimokančią sistemą bei daugiasluoksnio perceptrono dirbtinį neuroninį tinklą naudojantis pirmoje dalyje apžvelgtas metrikas, kaip įvestį. Siekiant pagerinti tinklo efektyvumą peržvelgiamos metrikos siekiant sudaryti glaustesnį duomenų rinkinį be perteklinių reikšmių. Tam padaryti vykdomi du eksperimentai: vieno metu duomenų rinkinys analizuojamas rankiniu būdų, o kito - naudojant duomenų analizės programinę įrangą. Galiausiai naudojantis naujai sudarytais duomenų rinkiniais adaptuojami bei testuojami daugiasluoksnio perceptrono tipo dirbtiniai neuroniniai tinklai skirti atspėti programinės įrangos defektyvumą.

Raktiniai žodžiai: dirbtiniai neuroniniai tinklai, Halstead, McCabe, daugiasluoksnis perceptronas

Summary

The paper analyzes the viability of using an artificial neural network trained to predict software defects as a tool in software testing process.

The paper consists of four main parts. Firstly an an overview of code measures and their relation to software defect proneness is conducted. After collecting the data an experiment is conducted where the results of several learning machines are compared. Mainly using dataset described in the first part two models are trained and tested, one basen on a decision tree the other one on a multilayer perceptron artificial neural network. Next are data analysis and feature selection. Two types of data analyses are carried out, one manually by removing redundant and directly proportionate values from dataset, the other one is done automatically by using a data analysis tool Weka. Finally the newly formed datasets are used to traina and test modified versions of the multilayer perceptron artificial neural network and the results are compared.

Keywords: artificial neural network, Halstead, McCabe, multilayer perceptron

TURINYS

Įvadas

Smarkiai augant Continuous Delivery principų populiarumui auga poreikis testuoti daugiau per trumpesnį laiko tarpą. Tai kelią didelę problemą testuotojams, kurie nori padengti kuo daugiau programinės įrangos funkcionalumo testais kiekvienos trumpos iteracijos metu. Paprastas sprendimas šiai problemai būtų be abejo nuolat besisukantis automatinių testų paketas, tačiau automatiniais testais padengti programinį kodą 100% praktiškai neįmanoma. 100% padengimas įmanomas tik pagal kokią nors specifinę metriką, o vaikytis tikrojo 100% padengtumo testais pagal kiekvieną metriką ar funkcinę sritį prilygsta šviesos geičio vaikymuisi, kuo arčiau esame tikslo tuo daugiau pastangų ir resursų reikia norint pasistūmėti į priekį. Todėl, žinoma, vis dar išlieka poreikis rankiniam testavimui. Norint išleisti programinės įrangos versiją kuo dažniau ištestuoti visko kiekvieną kartą neįmanoma netgi kombinuojant automatinius bei rankinius testus. Taip iškyla dar viena, mažesnė, problema: kaip efektyviai pasirinkti testavimo sritis kiekvienai naujai programos laidai (angl. release).

Alan M. Davis [Dav95] 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.

Pagrindiniai kriteriai pagal kuriuos dabar įparstai prioritetizuojami programinės įrangos moduliai testavimui yra: paskutinis modulio testavimo laikas, kuo ilgensį laiko tarpą netestuota tuo aukštesnę pirmenybę modulis įgauna, modulio naujumas, visiškai nauji programinės įrangos moduliai privalo būti testuojami, modulio atnaujinimas, jei modulis išplėstas ar sumažintas funkcionalumo prasme jis turi aukštą pirmenybę būti testuojamas ir galiausiai praeityje defektyvūs moduliai, tai tokie moduliai kurie praeityje turėjo defektų ir šie buvo ištaisyti.

Dažnai turint mažai laiko testuoti, modulių prioritetiavimas sueikvoja nemažą dalį laiko, kurį būtų galima skirti testavimui, taip pat dažnai tenka palikti daug aukšto prioriteto modulių nepilnai ištestuotus ar visai netestuotus. Tai didina programinės įrangos riziką būti nekokybiška. Ypač problematiška situacija, kai defektai lieka nesurasti keletą laidų (angl. release) ir po to tenka aiškintis ar defektas reiškia įvykusią regresiją ar jis visuomet egzistavo testuotame modulyje.

Šiame darbe aprašomas tyrimas skirtas patikrinti ar įmanoma ir ar tai efektyvu naudoti duomenis surinktus apie programinės įrangos laidą, kaip įvestį į dirbtinį neuroninį tinklą apmokytą nuspėti modulio defektyvumui, taip iš esmės automatizuojant modulių prioritetizavimo testavimo ciklui procesą.

Tyrimui atlikti naudojama programinė įranga Weka skirta duomenų analizei bei darbui su duomenimis, Python programavimo kalba, Keras bei Scikit-Learn python bibliotekų paketai skirti darbui su duomenimis, bei panaudoti sprendimų medžio bei daugiasluoksnio perceptrono dirbtinio neuroninio tinklo modeliams aprašyti.

Darbo tikslas: Patikrinti daugiasluoksnio perceptrono neuroninio tinklo, kaip įrankio programinės įrangos defektyvumui nuspėti, efektyvumą

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 sistemos ir daugiasluoksnio perceprtono tipo dirbtinio neuroninio tinklo naudojant surinktus duomenis.
3. Atlikti duomenų analizę duomenų rinkiniui siekiant pašalinti perteklines reikšmes.
4. Palyginti daugiasluoksnio perceptrono tipo dirbtinio neuroninio tinklo efektyvumą naudojant tą patį duomenų rinkinį jam pritaikius skirtingus atributų atrinkimo metodus.

Atsižvelgiant į užduotis darbas išskirstytais skyriais bei poskyriais perteikiančiais tyrimo vykdymo eigą:

1-ame skyriuje bus apžvelgiama keletas programinės įrangos matavimo būdų stengiantis rasti ar pagrįsti ryšį su programinės įrangos defektyvumu.

2-ame skyriuje programinės įrangos metrikos turinčios ryšį su programų defektyvumu apdorojomos keletu skirtingų besimokančių sistemų ir palyginami jų tikslumo rezultatai.

3-iame skyriuje duomenų rinkiniams atliekami duomenų atrinkimo procesai siekiant pašalinti besimokančioms sistemoms efekto neturinčius duomenis.

4-ame skyriuje pateikiami rezultatai naujų besimokančių sistemų pritaikytų dirbti su atnaujintais duomenų rinkiniais bei palyginami jų rezultatai.

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 nuspręsti kurios programinės įrangos kodo vietos, ar programų paketo moduliai turi didžiausią riziką būti defektyvūs. Į 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 ištestuoti visiems šiems moduliams ir kaip tuomet pasirinkti atitinkamus progaminės įrangos modulius testavimui, kad nešvaistydami laiko padarytumėme didžiausią įmanomą įtaką programinės įrangos kokybės užtikrinimui.

Visų pirma reikėtų pasirinkti detalesnę programinės įrangos metrikų sistemą nei, „naujumas“ ar „prieš tai buvusių defektų kiekis“, kuri tiesiogiai atspindėtų būsimą (ar esamą) programinės įrangos defektyvumą. Reikia surinkti sąlyginai lengvai surenkamas programinės įrangos metrikas, kurios suteiktų detalią informaciją apie programinės įrangos būklę ir turėtų ryšį su šios defektyvumu. Ir iš tiesų, būtų galima teigti, jog programinės kokybės metrikos, kurias būtų įmanoma tiesiogiai susieti su programinio kodo defektyvumu jau egzituoja. Arčiausiai tokių metrikų yra programino kodo sudėtingumo metrikos. Be abejo yra ne vienas būdas matuoti programos kodo sudėtingumą, tačiau, šiam tyrimui pasirinktos dvi, galima būtų sakyti, populiariausios metrikos kodo sudėtingumui atvaizduoti: Maurice H. Halstead aprašytosios 1977-aisiais [Hal77] bei Thomas J. McCabe aprašytosios 1976-aisiais [McC76]. Net ir patys autoriai savo publikacijose teigia, jog aukštas kodo sudėtingumas veda prie defektyvaus kodo.

Ryšys tarp kodo sudėtingumo bei defektų buvimo jame, 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 jame koreliaciją [SH99]. 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 [hCh13].

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 sąlyginai tiksliai parodo kodo defektyvumą.

Šiame dokumente aprašomame tyrime bus naudojamos visos Halstead ir McCabe metrikos kartu, bei keletas šių atributų poaibių susiaurintų tiek rankiniu būdų tiek duomenų analizės įrankiais siekiant optimizuoti duomenų rinkinius pašalinant perteklines reikšmes.

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 sąlygninai „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 sakinys su dviem salygomis sudarytų grafą kurio ciklomatinis sudėtingmas būtų lygus 3 ir t.t. [McC76].

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

Matematiškai ciklomatinis sudėtingumas M apskaičiuojamas formule

$$M = E - N + 2P.$$

Kur šios metrikos atvaizduojamos taip:

E = grafo kraštinių kiekis,

N = grafo viršūnių kiekis,

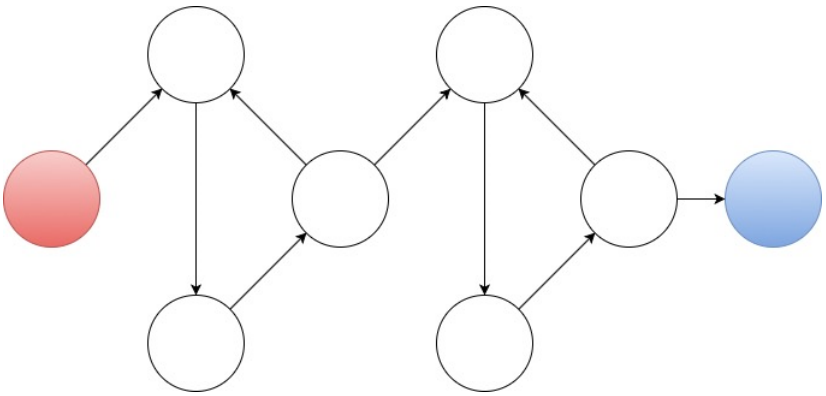
P = apjungtų komponentų kiekis.

Dar cikomatinis 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 1 pav. 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 viršūnių, kurie kartu sudaro vieną apjungtąjį komponentą, tuomet tokios programos ciklomatinis sudėtingumas būtų $9 - 8 + 2 * 1 = 3$.



1 pav. Programinio kodo atvaizdavimas grafu

McCabe taip pat apibėžė ir keletą kitų metrikų skirtų pamatuoti progaminio kodo sudėtingumą, viena tokių metrikų - esminis sudėtingumas. Esminis sudėtingumas - metrika parodanti programinės įrangos sudėtingumą pašalinant (suskaidant) logines struktūras, grafe vaizduojamas, kaip turinčias vieną viršūnę įvesčiai bei vieną rezultatams. Tokio tipo struktūrų pavyzdžiai būtų if-then-else sąlyginių sakinių dariniai ar while tipo ciklas.

Tarkime programinis kodas su dideliu ciklomatiniu sudėtingumu yra sunkiau ištestuojamas, tačiau esant nedideliam esminiam sudėtingumui reiškia, jog mes šį programinį kodą galime suskaidyti į mažesnius vienetus, kurie yra lengviau skaitomi ir testuojami atskirai. Didelis esminis sudėtingumas parodo, kad programinį kodą yra sunkiau skaityti bei prižiūrėti, t.y. programinis kodas yra žemesnės kokybės bei kyla didesnė defektyvumo rizika.

Thomas J. McCabe 1989-aisiais išpletė savo sudėtingumo metrikų publikaciją kartu su Charles W. Butler. Čia publikacijoje apibrėžiama nauja sudėtingumo metrika - dizaino sudėtingumas. Dizaino sudėtingumas tai ciklomatinis sudėtingumas redukuotam programos kodo grafui. Grafas redukuojamas siekiant pašalinti sudėtingus darinius kurie nedaro įtakos dizaino moduliams. Dizaino sudėtingumas parodo kodo modulio sąveikos su kitais moduliais sudėtingumą [MB89]. Metrika padeda atskirti tarp kodo modulių kurių buvimas programoje smarkiai padidina šios sudėtingumą ir tarp modulių, kurie paprasčiausiai savo viduje turi sudėtingos logikos elementų.

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 [Hal77].

Halstead metrikos paremtos kodo įsivaizdavimo kaip įvykių seką, kur kiekvienas įvykis yra operatorius arba operandas. Tuomet skaičiuojami šiais matavimo vienetais:

n_1 - unikalių operatorių kiekis,

n_2 - unikalių operandų kiekis,

N_1 - visų operatorių kiekis,

N_2 - visų operandų kiekis.

Halstead publikacijoje aprašomos šios metrikos apskaičiuojamos naudojant operandų kiekį, bei kitas kodo metrikas:

programos ilgis N,

programos žodynas n,

programos apimtis V,

programos sudėtingumas D,

programos lygis L,

programos pastangos E,

programos implementavimo laikas T,

programos defektų kiekis B.

Programos dydis N, dar vadinamas programos ilgiu tai visų operatorių ir operandų programoje kiekis ir apskaičiuojamas formule

$$N = N_1 + N_2.$$

Programos žodynas n, tai visų unikalių operatorių bei operandų kiekis programoje, kuris apskaičiuojamas formule

$$n = n_1 + n_2.$$

Programos apimtis V, tai programos informacinis turinys apskaičiuotas matematiškai. Ši metrika glaudžiai siejama su programos dydžiu bei žodynu ir apskaičiuojama formule

$$V = N * log_2 n.$$

Halstead programos apimtis matuoja operacijų bei operatorių kiekį aprašytame algoritme. Statistiškai vienos funkcijos apimtis turėtų būti tarp 20 ir 1000. Vienos eilutės funkcijos be parametrų apimtis apytiksliai 20, o 1000 ir didesnė apimtis rodo, kad matuojama funkcija daro per daug.

Programos sudėtingumas D, dar vadinamas polinkiu į defektus, tai reikšmė parodanti, kaip sunku skaityti ar modifikuoti programos kodą. Ši metrika yra proporcinga unikalių operatorių kiekiui ir apskaičiuojama formule

$$D = \frac{n_1}{2} * \frac{N_1}{n_2}.$$

Progrmos lygis L iš esmės metrika atvirkščia polinkiui į defektus. Kitaip tariant aukšto lygio programa yra atsparesnė defektams, nei žemo lygio. Ši metrika apskaičiuojama formule

$$L = \frac{1}{D}.$$

Programos pastangos E, tai metrika parodanti kaip sunku, t.y. kiek pastangų reikalauja, kodo skaitymas ar implementavimas. Metrika proporcinga programos apimčiai bei sudėtingumo metrikai ir apskaičiuojama formule

$$E = D * V.$$

Programos implementavimo laikas T, tai metrika parodanti kiek laiko užtruks suprasti ir

implementuoti programą. Metrika yra proporcinga su programos pastangomis. Metrika gali būti kalibruojama priklausomai nuo situacijos, tačiau Halstead rekomenduoja daliklį parinkti 18, taip gaunant laiką sekundėmis. Metrika apskaičiuojama formule

$$T = \frac{E}{18}.$$

Programos defektų kiekis B, tai metrika kuri apytiksliai pasako kiek defektų bus tam tikros implementacijos programoje. Pasak Halstead defektų kiekis C programavimo kalba parašytame faile turėtų būti apie 2, tačiau statistiškai tas kiekis būna žymiai didesnis. Defektų kiekis programoje tai metrika proporcinga programos pastangoms ir apskaičiojama formule

$$B = \frac{E^{\frac{2}{3}}}{3000}.$$

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 sistemą

Turint aiškiai apibrėžtas metrikas programinės įrangos kodui apibūdinti galima būtų vesti žurnalą kiekvienam programinės įrangos moduliui ir kiekvieną kartą renkantis programinės įrangos sritis, kurias reikia testuoti, palyginti esamos kodo laidos (angl. release) metrikas su žurnale esančiomis taip nusprendžiant ar išanalizuotą sriti reikia testuoti dabar ar sričiai priskirti žemesnę pirmenybę. Tačiau toks procesas atliekamas rankiniu būdu ne tik, kad ne taupo laiko, tačiau yra ir smarkiai neefektyvus bei prideda papildomų rūpesčių testuotojams sprendžiantiems testų prioritetus.

Jeigu pažvelgsime į turimą problemą iš kitos pusės, tuomet turime duomenų rinkinį, bei vieną iš dviejų reikšmių galimų priskirti tam duomenų rinkiniui: defektyvu arba ne defektyvu. Tai labai primena binarinio klasifikavimo problemą, kuri dažnai sprendžiama automatizuojant ir viską aprašant besimokančia sistema, dažniausiai dirbtiniu neuroniniu tinklu pvz. daugiasluoksniu perceptronu. Šiame tyrime bus naudojamos dviejų tipų besimokančios sistemos: sprendimų medžio spėjamasis modelis bei daugiasluoksniu perceptronu aprašytas dirbtinis neuroninis tinklas.

Apmokyti šias besimokančiasias sistemas 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 sistemai 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 kiekio 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 nuspręsta panaudoti keletą jau surinktų duomenų rinkinių. Siekiant surinkti pakankamą kiekį prasmingų duomenų (500-1000 skirtingų testuotų programinės įrangos versijų (angl. build)) šio tyrimo kontekste užimtų per daug laiko. Tyrime bus naudojami „CM1“ bei „PC1“ duomenų rinkiniai iš „Promise“ duomenų rinkinių duomenų bazės [SSM05]. Tiek „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 užfiksuotas vienas ar daugiau defektų bei 93.06% duomenų eilučių atspindinčių, kai nebuvo užfiksuota 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 užfiksuotas vienas ar daugiau defektų bei 9.83% duomenų eilučių atspindinčių, kai nebuvo užfiksuota defektų.

2.1. Sprendimų medžio metodu paremta besimokanti sistema

Pirmasis prediktyvusis modelis paremtas sprendimų medžio klasifikatoriumi. Toks modelis yra laisvai suprantamas ir skaitomas net ir paprastam žmogui. Vienas tokių modelių sugeneruotas naudojant PC1 duomenų rinkinio poaibį pavaizduotas 8 pav. 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 medį grafiškai pačiam galima pereiti per viršūnes tikrinant duomenų rinkinio eilutės grąžinamą rezultatą. Taip pat palyginus su kitais prediktyviaisiais modeliais sprendimo medžiai reikalauja salyginai nedidelio kiekio duomenų pertvarkymo prieš naudojant juos šiame modelyje. Iki tam tikro laipsnio smarkiai išsiskiriančios reikšmės ar trūkstami 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 sistemos 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).

Įprastai sprendimų medžio prediktyvusis modelis naudojamas salyginai paprastų duomenų rinkinių problemoms spręsti. Žemo lygio klasifikatoriams bei prediktyviesiems modeliams, kaip

Iriso gėlės skirtingų rūšių klasifikavimo problema.

Šio tyrimo metu sprendimų medžio modelis ir klasifikatorius buvo aprašyti ir įgyvedinti Python kalba parašytu kodu naudojant medžio biblioteką pateiktą Scikit-learn [PVGM+11] python bibliotekų rinkinį skirtą duomenų rinkimui bei analizei.

Rezultatai taip pat buvo apskaičiuoti keletu scikit-learn pateiktų bibliotekų, kurių viduje jie apskaičiuojami keletu formulių:

$$Precision = \frac{\Sigma TikrasPozityvas}{\Sigma TikrasPozityvas + \Sigma NetikrasPozityvas},$$

$$Accuracy = \frac{\Sigma TikrasPozityvas + \Sigma TikrasNegatyvas}{\Sigma Pozityvai + \Sigma Negatyvai},$$

$$Recall = \frac{\Sigma TikrasPozityvas}{\Sigma TikrasPozityvas + \Sigma NetikrasNegatyvas},$$

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. Accuracy metrikos atveju Pozityvai aprėpia tiek teisingai, tiek neteisingai nuspėtas pozityvias reikšmes, o negatyvai - negatyvias.

Besimokančių sistemų 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 sistemos rezultatai

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

Lentelė 1 atvaizduoja accuracy, percision bei recall reikšmes gautas iš sprendimų medžiu paremto modelio. Modeliui apmokyti ir testuoti abiem atvejais buvo panaudotas visas duomenų rinkinys su 21 atributu, kur 70% atsitiktinai atrinktų reikšmių buvo panaudotos, kaip mokomieji duomenys, o kiti 30%, kaip testiniai.

Sprenimo medžiu paremtas prediktyvus modelis panašu, jog nėra optimaliausias sprendimas programos modulių defektyvumui nuspėti. Tiek su CM1, tiek su PC1 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 nelygų duomenų pasiskirstymą, kodėl čia išvydome „permokymo“ reiškinį. Tačiau pirmiausia reikėtų patikrinti ar su tokiais pat, ne modifikuotais, duomenimis galima gauti gerensius rezultatus naudojant kitokį klasifikatorių.

2.2. Daugiasluoksnis perceptronas

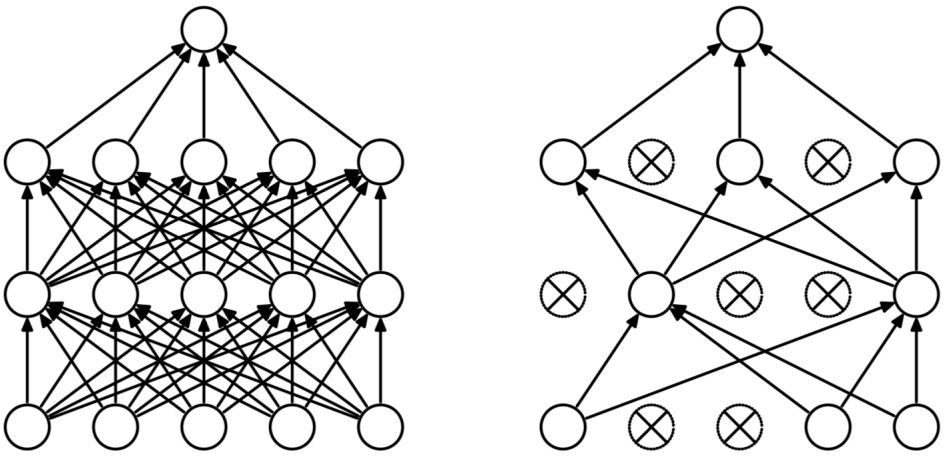
Antrasis prediktyvusis modelis tyrimo metu buvo aprašytas daugiasluoksio perceptrono klasifikatoriumi. Daugiasluoksnis perceptronas yra tiesinio tipo dirbtinis neuroninis tinklas, kurio ryšiai tarp viršūnių (vaizduojant modelį grafu) yra nukreipti į vieną pusę, t.y. ne rekurentiniai. Ir duomenys tiesiškai keliauja nuo įvesties sluoksnio per paslėptuosius sluoksnius iki gaunamas rezultatas.

Kadangi nėra taisyklių ar geriausių praktikų daugiasluoksnių perceptronų, ar kitų dirbtinių neuroninių tinklų aprašymui pasirinkta ieškoti salyginai efektyviausio tinklo modelio pasirinktiems duomenų rinkiniams atliekant keletą eksperimentų. Visų pirma pasirinkta vieno paslėpto sluoksnio modelio struktūra bei pasirinktas 100 epochų kiekis modelio treniravimui. Epocha - viena iteracija per visą duomenų rinkinį. Taip treniruotas modelis turėjo accuracy reikšmę 89.76% treniravimui bei testavimui naudojant CM1 duomenų rinkinį. Toliau toks pat eksperimentas atlikas su 1000 epochų treniravimo ciklu, tačiau galutinis accuracy rezultatas grąžintas nepasikeitęs. Toliau modelis buvo atnaujintas paslėptąjį sluoksnį sumažinant trečdaliu, t.y. iki 14 viršūnių. Treniruojant tokį modelį tiek 100 epochų, tiek 1000 epochų buvo gautas tas pat accuracy rezultatas - 89.98%. Remiantis šiais rezultatais pasirinktas daugiasluoksnio perceptrono modelis su dviem paslėptais sluoksniais siekiant salyginai nesudėtingo, tačiau efektyvaus dirtinio neuroninio tinklo.

2 lentelė. Eksperimentų su daugiasluoksnio perceptrono modeliu rezultatai

	Epochų kiekis	Accuracy
Vienas sluoksnis (21 viršūnė)	100	89.76%
Vienas sluoksnis (21 viršūnė)	1000	89.76%
Vienas sluoksnis (14 viršūnių)	100	89.98%
Vienas sluoksnis (14 viršūnių)	1000	89.98%
Du sluoksniai (14 ir 7 viršūnės)	100	90.16%

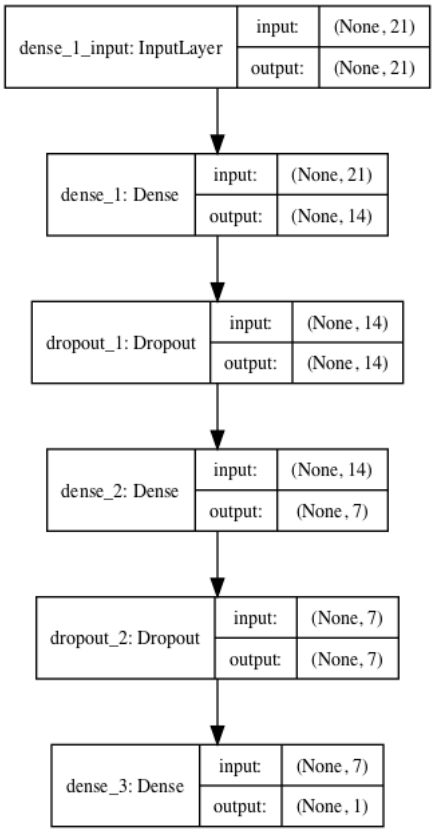
Siekiant didelio efektyvumo šiame modelyje pridėti du paslėpti sluoksniai su vis mažesniu kiekiu neuronų, taip bemokant modelį priverčiant jį atskirti svarbiausias reikšmes darančias didžiausią įtaką galutiniam rezultatui. Pasirinktos reikšmės 14 viršūnių pirmam paslėptam sluoksniui bei 7 viršūbės antrajam. Taip pat po kiekvieno iš paslėptųjų sluoksnių įterptas išmetimo sluoksnis, kuris sumažina praeitame sluoksnyje sukurtų ryšių kiekį taip mažinant tinklo permokymo riziką [SHKSS14]. Tokiu atveju dirbtinis neuroninis tinklas tampa panašenis į tokį, vaizduojamą modeliu b 2 pav..



(a) Standard Neural Net (b) After applying dropout.

2 pav. „Dropout“ sluoksnių efektas daugiasluoksniam perceptronui.

Toliau šiame tyrime naudojamas daugiasluoksnio perceptrono modelis atrodys taip, kaip vaizduojama 3 pav., kur įvesties (angl. input) sluoksnyje turime 21 atributą, kaip ir duomenų rinkinyje, toliau du užslėptus sluoksnius, su trečdaliu mažiau įvesčių siekiant sukurti glaudesnius ir prasmingenius ryšius tarp modelio viršūnių, kas kart apmažinant ryšių kiekį siekiant išvengti modelio permokymo.



3 pav. Daugiasluoksnio perceptrono struktūra

Daugiasluoksnis perceptronas taip pat dažnai naudojamas paprastų duomenų rinkinių, atvaizduojamų lentele, analizei. Šio tyrimo kontekste toks dirbtinis neuroninis tinklas tinkamiausias, nes daugeliu kitų atvejų, pavyzdžiui konvoliucinio neuroninio tinklo, kuris savo struktūra yra sudėtingesnis bei daugeliu atvejų efektyvesnis, reikėtų adaptuoti bei transformuoti duomenų rinkinį, to buvo siekiama išvengti kuo tikslesniam palyginimui su sprendimų medžio modeliu.

Daugiasluoksnis perceptronas buvo aprašytas Python programavimo kalbą pasinaudojus „Tensorflow“ paremtu python bibliotekų rinkiniu skirtu darbui su sekvenciniais neuroniniais tinklais - „Keras“ [Cho+15]. Kaip ir sprendimų medžio atveju buvo skaičiuojamos reikšmės matuojančios dirbtinio neuroninio tinklo accuracy, precision ir recall reikšmes.

Dirbtiniam neuroniniam tinklui apmokymas ir testavimas vyko du kart. Vieną kartą naudojant PC1 duomenų rinkinį, kitą kartą naudojant CM1. Abiejais atvejais buvo panaudotas pilnas duomenų rinkinys su 21 atributu, kur 30% duomenų buvo skiriama apmokyto modelio testavimui, o 70% duomenų buvo skirta modelio apmokymui.

3 lentelė. Daugiasluoksnio perceptrono rezultatai

Duomenų rinkinys	accuracy	precision	recall
PC1	93.06%	92.98%	92.94%
CM1	90.16%	89.88%	89.86%

Taigi, lentelė 3 atvaizduoja abiejų eksperimentų su pilnų duomenų rinkiniu rezultatus, ir iš karto galime pamatyti, jog jau vien accuracy reikšmė keletu procentų aukštesnė, nei sprendimų medžio vaizduojamo lentelėje 1, abejais atvejais, Tačiau abi pasirinktosios metrikos naudojamos šiame tyrime turi nemažai tiesiogiai vieni kitiems proporcingų reikšmių, kurias pašalinus būtų galima dar labiau padidinti dirbtinio neuroninio tinklo efektyvumą sukuriant glaudesnius ryšius tarp mažesnio atributų skaičiaus.

3. Halstead bei McCabe sudėtingumo metrikų analizė dirbtinių neuroninių tinklų kontekste

Neatrūšiuoti duomenys skirti apmokyti dirbtinius neuroninius tinklus, sudaryti iš didelės įvairovės atributų, kurių tik dalis daro įtaką. Atributų atrinkimas (angl. feature selection) yra dažna praktika skirta padidinti dirbtinio neuroninio tinklo efektyvumą bei išvengti tinklo permokymo.

Turint platų dirbtinį neuroninį tinklą, t.y. tinklą priimantį didelį kiekį skirtingų atributų, ypač, tokį kuriame nėra daug paslėptų sluoksnių, sudaromos salygos apmokymo metu tinklą išmokyti prisiminti, vietoj to, ko iš jo tikimasi - generalizuoti. Toks dirbtinis neuroninis tinklas galiausiai gali puikiu tikslumu grąžinti spėjimą duomenims su kuriais buvo apmokytas, tačiau atsiradus naujiems duomenims atlikti duomenų klasifikavimo jau nebesugeba.

Atributų atrinkimas tai procesas kurio metu atrenkami konkrečiam dirbtinio neuroninio tinklo modeliui svarbūs duomenų rinkiniai. Atributų atrinkimas gali pagelbėti konstruojant dirbtinio neuroninio tinklo modelį. Įvykdžius atributų atrinkimą modeliu gaunamas toks pat ar potencialiai geresnis rezultatas su daug mažiau duomenų.

Atributų atrinkimo metodai gali būti naudojami atrinkti nereikalingus, perteklinius bei beprasmius duomenis iš duomenų rinkinio, kurie neprisideda prie modelio tikslumo gerinimo, o gali ir būti priežastimi dėl žemo modelio tikslumo.

Mažesnis atributų kiekis taip pat padeda sudaryti mažesnius bei ne tokius sudėtingus modelius, kurios lengviau prižiūrėti bei suprasti.

Automatiniam atributų atrinkimui yra keletas pagrindinių algoritmų: filtro metodai, grupių metodai ir vidiniai metodai.

Filtro atributų atrinkimo algoritmai (angl. Filter Methods) vykdo statistinę duomenų analizę, po kurios kiekvienam iš atributų priskiriamas svoris. Tuomet visi atributai rūšiuojami pagal jų svorio rezultatus ir priklausomai nuo ribinio svorio išimami arba paliekami duomenų rinkinyje. Keletas tokio algoritmo pavyzdžių būtų Information Gain algoritmas bei Correlation coefficiet score algoritmas.

Grupių atributų atrinkimo algoritmai (angl. Wrapper methods) atributų atrinkimą vykdo kaip paieškos uždavinį, kur daugybė skirtingų atributų kombinacijų parenkamos, įvertinamos bei lyginamos su kitomis. Perdiktyvusis modelis naudojamas kiekvieno duomenų poaibio vertinimui ir atributų rinkinio įverinimo rezultatas sutampa su modelio accuracy metrika. Tokio algoritmo pavyzdys - rekursyvus atributų atmetimo algoritmas.

Vidiniai atributų atrinkimo algoritmai (angl. Embedded methods) atlieka duomenų analizę ir atributų atrinkimą modelio kūrimo metu. Dažniausiai pasitaikantys vidiniai atributų atrinkimo metodai yra reguliarizavimo metodai.

Reguliarizavimo metodai dar dažnai vadinami drausmės metodais, tai toks metodas, kuris modelio apmokymo metu prideda papildomų apribojimų prediktyviajam algoritmui. Toks procesas padeda sugeneruoti mažesnį sudėtingumą turintį modelį. Keletas tokių algoritmų pavyzdžių yra LASSO bei Ridge Regression.

Tęsiant tyrimą bus vykdomi du skirtingi atributų atrinkimo metodai. Pradedant su pilnu duomenų rinkiniu susidedančiu iš 21 atributo rankinius būdu bus atrinkti atributai, kurie tiesiogiai susiję su kitais, t.y. jei Halstead metrikos išvedamos naudojant unikalių operandų bei operatorių kiekius, šie abu atributai bus išimami iš duomenų rinkinio, o palikti tik iš jų abiejų išvesti duomenys.

Kitas duomenų atrinkimo metodas bus atliktas automatiškai, naudojant duomenų analizės įrankį „Weka“. Išbandžius keletą atrinkimo metodų bei filtro atributų atrinkimo algoritmų bus sudaryti nauji duomenų rinkiniai - senojo poaibiai.

3.1. Duomenų analizė rankiniu būdu, ieškant atributų, kurie yra tiesiogiai proporcingi kitiems atributams

Duomenų atrinkimas buvo pradėtas nuo Halstead reikšmių, kurios nors ir pateikia mums empirines metrikas kodui yra išvestinės, t.y. keturios metrikos apibūdinančios operandų kiekį kode yra kombinuojamos tarpusavyje gauti kitoms todėl jos yra tiesiogiai proporcingos ir perteklinės dirbtinio neuroninio tinklo mokymo kontekste, todėl gali vesti permokymo link. Taip pat išimtos reikšmės tiesiogiai darančios įtaką kodo skaitomumui, tačiau neturinčios jokios esminės įtakos kodo vykdyme, kaip komentarų eilučių kiekis ar tuščių eilučių kiekis.

Pašalinta n - Halstead visų operatorių bei operandų suma, metrika iš esmės parodanti tą pat, kaip ir eilučių kode kiekis, todėl keletas tokio pat tipo metrikų sukurtų perteklinės informacijos dėl šios priežasties išimtos ir kitos kodo metrikos matuojančios tiesiog kodo apimtį, kaip lOCode - kodo eilučių kiekis, lOComment - komentarų eilučių kiekis, lOBlank - tuščių eilučių kiekis bei lOCodeAndComment - bendras komentarų bei kodo eilučių kiekis.

Pašalintos operandų metrikos: uniq_Op, uniq_Opnd, total_Op bei total_Opnd, nes dirbtinio neuroninio tinklo konrtekste jos nesuteikia daug vertės ir tik gali privesti prie tinklo permokymo, nes yra perteklinės reikšmės, tiesiogiai proporcingos nuo kitų ir naudojamos apskaičiuoti visas Halstead reikšmes.

Taip pat išimtos visos žemesnio lygio reikšmės skirtos apskaičiuoti tiek McCabe metrikas tiek papildomai pridėtos duomenų rinkinyje. Taip paliekant 10-ies atributų duomenų rinkinį sudarytą vien iš išvestinių reikšmių, kurios buvo tiesiogiai proporcingos nuo išimtųjų, tačiau tarpusavyje tokio ryšio neturi.

Po šio duomenų atrinkimo metodo duomenų rinkinio atributai atrodo 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.

v - Skaitinė reikšmė. Halstead programos apimtis.

d - Skaitinė reikšmė Halstead sudėtingumas.

i - Skaitinė reikšmė. Halstead protingo turinio metrika.

e - Skaitinė reikšmė. Halstead pastangos.

b - Skaitinė reikšmė. Halstead defektų kiekio spėjimas.

t - Skaitinė reikšmė. Halstead laiko suprogramuoti metrika.

defects - Binarinė reikšmė. True arba False.

3.2. Duomenų analizė naudojant įrankį Weka

Weka tai darbo su dideliais duomenų rinkiniais įrankis, savyje turintis sistemų mokymosi algoritmų duomenų analizės užtuotims vykdyti. Šie algoritmai gali būti pritaikyti tiesiai duomenų rinkinius atidarius per Weka grafinę sąsają, arba iškviesti Java kodu ir apdoroti naudojantis Weka bibliotekomis. Šio tyrimo metu buvo naudotasi Weka grafine sąsaja.

Duomenų analizės įrankis Weka [HFHPRW09] turi daug su duomenų analize susijusių paskirčių, kaip vizualus duomenų lentelių atvaizdavimas, duomenų atrinkimas ir kt. Tyrimo kontekste bus panauodoti keletas Weka viduje esančių atributų atrinkimo algoritmų, kuriuos apžvelgus bus atrinkti nauji, sumažinti, duomenų rinkinai.

Iš leidžiamų pasirinkti atributų įvertinimo algoritmų šiam tyrimui tinkamiausi du: „CorrelationAttributeEval“ algoritmas kiekvieną reiškmę imantis kaip „pagrindinę“ ir lygindamas ją su kitomis grąžina koreliacijos tarp tos reikšmės bei galutinio reultato koeficientą, bei „InfoGainAttribute“ algoritmas palygindamas viso duomenų rinkinio atributų reikšmes grąžina kiekvienio atributo koeficientą reiškiantį informacijos tame atribute svarbą.

Panaudojus abu algoritmus buvo gauti tokie rezultatai:

4 lentelė. Duomenų rinkinių analizės įrankiu Weka rezultatai I

	loc	v(g)	ev(g)	iv(g)	n	v	l	d
CorrelationAttributeEval	0.2465	0.1668	0.105	0.2026	0.214	0.2066	0.1326	0.1682
InfoGainEval	0.0457	0.0274	0	0.0431	0.0426	0.0452	0.0291	0.0306

5 lentelė. Duomenų rinkinių analizės įrankiu Weka rezultatai II

i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndComment
0.2678	0.0978	0.2152	0.0978	0.063	0.3016	0.1735	0.0475
0.0461	0.0391	0.0493	0.0391	0.02	0.0559	0.0398	0

6 lentelė. Duomenų rinkinių analizės įrankiu Weka rezultatai III

uniq_op	uniq_opnd	total_op	total_opnd	branchCount
0.2493	0.2616	0.2141	0.212	0.1697
0.0493	0.0499	0.0432	0.0362	0.0281

Koreliacijos algoritmui renkantis reikšmes iki 0.2, o informacijos svarbos algoritmui iki 0.04 galima atrinkti ir palyginti reikšmes. Abejais atvejais gaunamas gana panašus duomenų rinkinys. Kuriuos apjungus gaunami 10 svarbiausių atributų.

Galutiniai duomenų rinkiniai bus sudaryti iš viso iš 11 stulpelių, kur paskutiniame saugomi rezultatai, o kti 10 yra pradinio duomenų rinkinio poaibis ir atrodo taip:

loc – Skaitinė reikšmė. McCabe eilučių kode kiekis.

iv(g) – Skaitinė reikšmė. McCabe dizaino sudėtingumas.

n – Skaitinė reikšmė. Halstead visų operatoriu bei operandų suma (programos dydis).

v – Skaitinė reikšmė. Halstead programos apimtis.

i – Skaitinė reikšmė. Halstead protingo turinio metrika.

b – Skaitinė reikšmė. Halstead defektų kiekio spėjimas.

lOComment – Skaitinė reikšmė. Halstead komentarų eilučių kiekis.

uniq_Op – Skaitinė reikšmė. Unikalių operatorių kiekis.

uniq_Opnd – Skaitinė reikšmė. Unikalių operandų kiekis.

total_Op – Skaitinė reikšmė. Visų operatorių kiekis.

defects – Binarinė reikšmė. True arba False.

Žinoma tokia analizė nėra visiškai tiksli ir smarkiai priklauso no to koks bus pasirinktas analizės algoritmas ir kaip stuktūrizuoti bei pasiskirstę duomenys. Kaip paaiškėjo po duomenų analizės, abu algoritmai didelę svarbą vis tiek teikia eilučių, komentarų metrikoms, kurios rankiniu būdu atrenkant atributus buvo pašalintos. Šių duomenų kontekste šios reikšmės turi didelę svarbą ir padės pagerinti dirbtinio neuroninio tinklo rezultatus, tačiau realybėje tarp jų ir kodo defektyvumo nėra jokio glaudaus ryšio.

Realiu atveju rekomenduotina naudoti tiek rankinio atrinkimo tiek algoritminio atrinkimo technikų kombinaciją ir atrinkti reikšmes rankiniu būdu jas dar išanalizuoti automatiškai arba atvirkščiai.

4. Dirbtinio neuroninio tinklo efektyvumo gerinimas naudojant atrinktus duomenis

Atlikti du eksperimentai su dviemis skirtingomis besimokančiomis sistemomis, viena paremta sprendimų medžiu, o kita paremta daugiasluoksnio percetrono modeliu atvaizduotu dirbtiniu neuroniniu tinklu. Gavus rezultatus pastebėta, jog su pilnu duomenų rinkinu sprendimų medžio sistemoje įvyksta persimokymas, o daugiasluoksnio perceptrono rezultatai, nors ir geresni, tačiau ne visai tenkina, nes, nors ir esant 90% tikimybei, jog neurininis tinklas programinės įrangos modulį, kaip defektyvų klasifikuos sėkmingai, egzistuoja 10% tikimybė, jog duomenys bus indentifikuoti klaidingai.

Rezultatams gerinti priimtas sprendimas atlikti atributų išrinkimą dviem būdais. Pirmasis būdas buvo skirtas išrinkti duomenis kurie tarpusavyje neturi koreliacijos ir yra visiškai individualios reikšmės palyginus su kitais atributais. Kadangi duomenų rinkiniai sudaryti pagal McCabe ir Halstead metrikas išimti pradiniai atributai iš kurių išvestos reikšmės, taip pašalinant beveik puse visų buvusių atributų tokių kaip unikalių operandų kiekis ar kodo eilučių kiekis. Antrasis būdas buvo įvykdytas automatiškai, duomenų rinkinį įkėlus į duomenų analizės programą ir pasirinkus analizės algoritmą, kurio rezultatas svarbos reikšmės kiekvienam iš atributų. Taip pasirenkant apytiksliai puse vsų buvusių atributų.

Iš viso buvo gauti keturis naujus duomenų rinkinus, kuriuos dėl aiškumo pavadinkime PC1_R bei CM1_R bei PC1_A bei CM1_A. Čia R reiškia, jog duomenų rinkiniai buvo analizuoti rankiniu metodu, o A reiškia, jog duomenų rinkinys buvo sugeneruotas automatiškai atrinkus bei atmetus neefektyvius ir įtakos nedarančius atributus.

Tęsiant tyrimą nebenaudosime sprendimų medžio metodo, nes vien su mažiau efektyviu, pertekliniu, duomenų rinkiniu dirbtinis neuroninis tinklas ne tik, kad nebuvo permokytas, tačiau ir apskritai grąžino labai gerus rezultatus. Tolesnis eksperimentas bus skirtas pagerinti jau egzistuojantį dirbtinio neuroninio tinklo efektyvumą, taip sumažinant riziką, jog programinės įrangos versija bus neteisingai įvertinta, kaip defektyvi ir įtraukta į testavimo ciklą.

Tiesiogiai daugiasluoksnio perceptrono tinklo modeliui pritaikyti naujų duomenų rinkinių negalima. Eksperimentui su pilnu duomenų rinkiniu sukurtas daugiasluoksnis perceptronas priima tik pilną duomenų rinkinį. t.y. duomenų eilutes sudarytas iš 21 atributo. Tokiu atveju reikia atnaujinti visą modelį.

Eksperimento kontekste, norint palyginti dvi reikšmes, jos turėtu turėti tą patį pradą, t.y. būti sugeneruotos to paties modelio. Šiuo atveju modelį reikia keisti, tačiau tik pirmąjį, įvesties, sluoksnį. Pakeitus šį sluoksnį ir pritaikius naujiesiems duomenų rinkiniams eksperimentas bus vykdomas su modeliu vaizduojamu 9 pav. esančiame priede nr. 2.

Didžiausia šio modelio problema kurią realiame gyvenime galima būtų ištaisyti, tačiau tyrimo kontekste reikia palikti modelyje esantį išsiplėtimą iš 10-ies į 14-a reikšmių tarp įvesties ir pirmojo paslėpto sluoknsių. Siekiant dar geresnio modelio efektyvumo antrajame sluoksnyje galima būtų sumažinti įvesčių kiekį iki 10 ar mažiau, tačiau tokiu atveju tai būtų jau visiškai kitoks modelis, ir rezultatų palyginimas tarp 21 atributo modelio bei modelio skirto 10 atributų būtų

neteisingas.

Pirma atnaujintas modelis buvo apmokytas ir ištestuotas naudojant duomenų rinkinius atrinktus rankiniu būdu, t.y. PC1_R bei CM1_R. Kaip ir su pilnu duomenų rinkiniu tas pat duomenų rinkinys skaidomas į dvi dalis: vieną skirtą apmokyti modelį, sudarančią 70% viso duomenų rinkinio, o kitą - jam testuoti sudarančio 30% viso duomenų rinkinio.

7 lentelė. Daugiasluoksnio perceptrono rezultatai, naudojant rankiniu būdu atrinktus atributus

Duomenų rinkinys	accuracy	precision	recall
PC1_R	93.54%	93.32%	93.27%
CM1_R	90.45%	90.39%	90.28%

Eksperimento rezultatai vaizduojami 7 lentelėje, kaip ir tikėtasi yra geresni, nei naudojant pilną duomenų rinkinį. Tačiau tikslumo pagerėjimas nėra akivaizdus ir siekia vos mažą dalelytę, galima sakyti rezultatai bene tokie pat, kaip pirmo eksperimento metu, kaip pavaizduoti 3 lentelėje.

Toliau kartojamas ekperimentas su kitais dviem duomenų rinkiniais PC1_A bei CM1_A. Čia taip pat duomenys dalijami į dvi dalis, 70% apimties poaibį skirtą modelio apmokymui bei 30% poaibį skirtą modelio testavimui.

8 lentelė. Daugiasluoksnio perceptrono rezultatai, naudojant atributus atrinktus automatiškai Weka programa

Duomenų rinkinys	accuracy	precision	recall
PC1_A	94.02%	93.86%	93.79%
CM1_A	91.05%	90.89%	90.78%

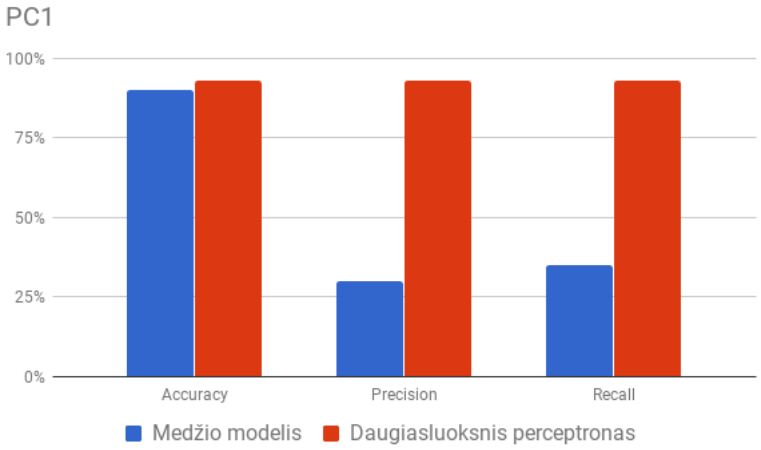
Antrojo eksperimento su atrinktų atributų duomenų rinkiniais rezultatai dar geresni, nei rankiniu būdu atrinktų duomenų rinkinių. Tačiau vėlgi nėra ekstremalaus pokyčio palyginus su pirmojo eksperimentu su daugiasluoksniu perceptrono rezultatais vaizduojamais 3 lentelėje. Galima drąsiai teigti, jog atributų išrinkimas, priklausomai nuo atrinkimui naudojamo metodo, padidins dirbtinio neuroninio modelio tikslumą atspėjant programinės įrangos defektus. Tačiau šių duomenų rinkinių kontekste dėl keleto dešimtųjų procento, kurių nebūtų galima pavadinti smarkiu pagerėjimu, investuoti daugiau laiko ar resursų nebūtų verta.

Rezultatai ir išvados

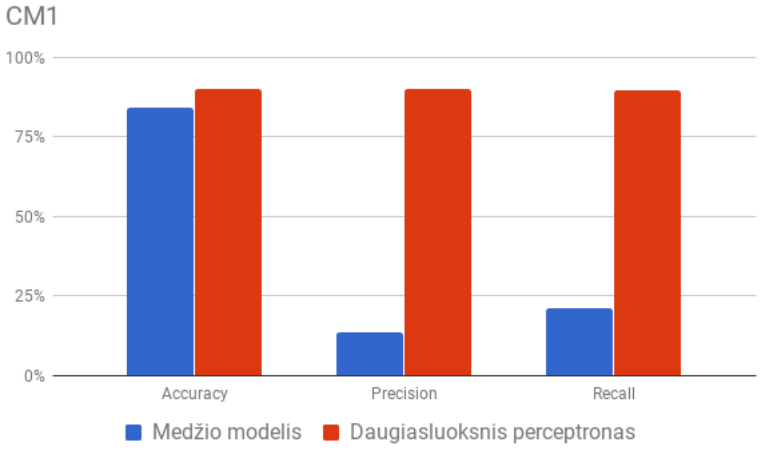
1. Išrinktas kodo metrikų rinkinys. Atlikus literatūros analizę pasirinktios Halstead bei McCabe kodo metrikos, matuojančios kodo sudėtingumą tiek logine prasme, McCabe kelių per grafu vaizduojamą kodą kiekis, tiek kodo skaitomumo prasme Halstead metrikose. Pasirinktos būtent šios dvi metrikos, nes šios metrikos gali būti surinktos ir paskaičiuotos statinės analizės metu, todėl jas salyginai lengva surinkti ir su jomis eksperimentuoti.

Esant dideliam kodo skaitomumo sudėtingumui ar loginiam sudėtingumui didėja defektų atsiradimo rizika. Nors ir yra abejojančių, ypač Halstead metrikų, sąryšiu su kodo defektyvumu, vis vien siūloma ne atsisakyti šių metrikų, o geriau metrikas parinkti pritaikius esamai situacijai.

2. Atlikti du eksperimentai. Su surinktomis kodo metrikomis bei apskaičiuotomis sudėtingumo reikšmėmis buvo apmokytos dvi besimokančios sistemos viena paremta sprendimų medžio modeliu, kita daugiasluoksnio perceptrono dirbtiniu neuroniniu tinklu. Accuracy metrikos visais atvejais, kaip vaizduojama 4 ir 5 pav., aukštos, tačiau palyginus gautuosius rezultatus matome, jog naudojantis abejais duomenų rinkiniais sprendimų medžio atveju įvyko persimokymo reiškinys, kai padavus naujų duomenų rinkinį modelis nebesugeba atspėti reikšmės.



4 pav. Rezultatų palyginimas su duomenų rinkinu PC1



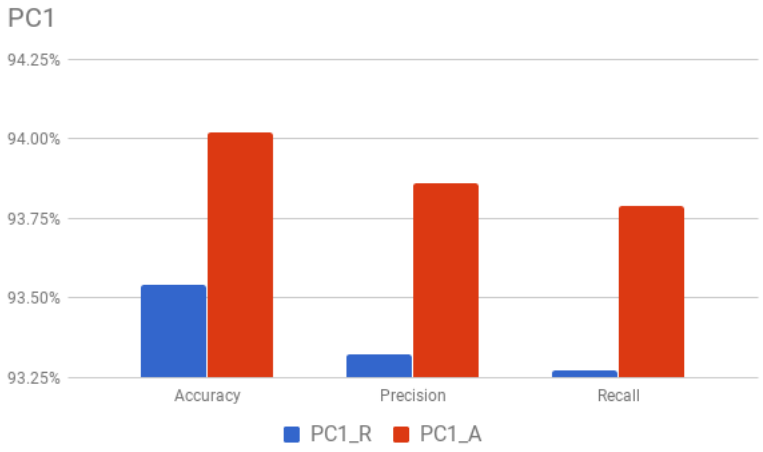
5 pav. Rezultatų palyginimas su duomenų rinkinu CM1

Iš rezultatų paaiškėjo, jog prendimų medžio modelis šiam uždaviniui spręsti nebuvo tinkamas. Dėl didelio atributų kiekio, šiuo atveju 21 atributas, įvyko modelio permokymas.

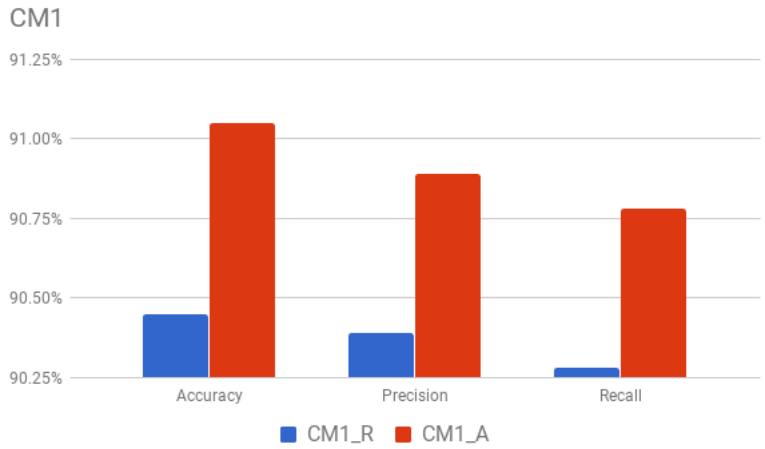
3. Atlikta duomenų analizė siekiant sumažinti perteklinių, tiesiogiai proporcinų bei kitų neigiamą įtaką dirbtiniam neuroniniam tinkkui galinčių padaryti reikšmių kiekį. Duomenims atrinkti naudoti du metodai: rankinis atrinkimas remiantis Halstead ir McCabe metrikų formulėmis, kai duomenų rinkinyje paliekamos tik išvestosios reikšmės, o metrikų iš kurių tos reikšmės išvedamos stulpeliai pašalinami bei duomenų atrinkimas naudojantis duomenų analizės įrankiu Weka, kai reikšmės paskirstomos pagal jų daromą įtaką galutiniam rezultatui.

Kombinuojant visas Halstead bei McCabe metrikas gauname daugybe panašių, bei tiesiogiai proporcingų vienų kitoms metrikų, kurios nors ir padeda susidaryti geresnį įspūdį apie kodo kokybę metrikas peržiūrint, tačiau tik apkrauna dirbtinį neuroninį tinklą pertekliniais duomenimis, kurie gali privesti prie persimokymo.

4. Atlikti du eksperimentai, naudojant šiek tiek pakeistą dirbtinį neuroninį tinklą, kuriame buvo atnaujintas įvesties sluoksnis, pritaikytas mažesniam atributų kiekiui. Eksperimento metu lyginti dirbtinio neuroninio tinklo mokymosi bei reikšmių nuspėjimo rezultatai naudojantis du modifikuotus duomenų rinkinius. Kur pirmuoju atveju, PC1_R bei CM1_R, iš duomenų rinkinio išimti visi atributai simbolizuojantys reikšmes iš kurių išvedamos Halstead bei McCabe reikšmės, ir paliktos tik išvestosios. Antruoju atveju, PC1_A bei CM1_A, duomenys atrinkti automatiškai, naudojant duomenų analizės įrankį Weka, pasirinkus algoritmą atributus paskirstantį pagal galutiniam rezultatui daromą įtaką.



6 pav. Rezultatų palyginimas su duomenų rinkinio PC1 poaibiais



7 pav. Rezultatų palyginimas su duomenų rinkinio CM1 poaibiais

Abejais atvejais tiek accuracy tiek precision bei recall savybės dirbtiniame neuroniniame tinkle pagerėjo turint kuruotą duomenų rinkinį, tačiau didelę įtaką darančio skirtumo tarp rankiniu būdu atrinktų atributų, bei išanalizuotų tam skirtu įrankiu nėra, kaip vaizduojama 6 ir 7 pav.. Vykdyti atributų atrinkimą yra verta ypač turint duomenų rinkinį turintį galybę perteklinių atributų, tačiau, geresnis sprendimas būtų užuot renkantis vieną iš metodų duomenų atrinkimą rankiniu būdų kombinuoti su duomenų analize tam specializuotais įrankiais, priklausomai nuo duomenų rinkinio.

Galiausiai, atlikus keletą eksperimentų, galima teigti, jog programinės įrangos testavimo prioritetų parinkimas renkant duomenis kodo statine analize bei juos panaudojant apmokant bei atliekant spėjimus dirbtiniu neuroniniu tinklu yra įgyvendinamas sprendimas. Toks procesas padės prioritetizuoti testuotinus programinės įrangos modulius, kai aukšto prioriteto modulių yra daug ir neįmanoma jų visų įtraukti į testavimo ciklą, taip smarkiai pagerinant kuriamos programinės įrangos kokybę.

Literatūra

[Cho+15] François Chollet ir k.t. Keras. https://keras.io, 2015.

[Dav95] Alan M. Davis. *201 Principles of Software Development*. McGraw-Hill, Inc., New York, NY, USA, 1995. ISBN: 0-07-015840-1.

[Hal77] Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977. ISBN: 0444002057.

[hCh13] Chen huei Chou. Article: metrics in evaluating software defects. *International Journal of Computer Applications*, 63(3):23–29, 2013. Full text available.

[HFHPRW09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann ir Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[MB89] Thomas J. McCabe ir Charles W. Butler. Design complexity measurement and testing. *Commun. ACM*, 32(12):1415–1425, 1989-12. ISSN: 0001-0782. DOI: 10.1145/76380.76382. URL: http://doi.acm.org/10.1145/76380.76382.

[McC76] Thomas J. McCabe. A complexity measure. *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, p. 407–, San Francisco, California, USA. IEEE Computer Society Press, 1976. URL: http://dl.acm.org/citation.cfm?id=800253.807712.

[PVGM+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel ir k.t. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SH99] Mark Schroeder ir Brian Henderson. A practical guide to object-oriented metrics. 1999.

[SHKSS14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever ir Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[SSM05] J. Sayyad Shirabad ir T.J. Menzies. The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. URL: http://promise.site.uottawa.ca/SERepository.

Sąvokų apibrėžimai

Continous delivery – programinės įrangos kūrimo metodika, kurios metu komandos dirba mažomis iteracijomis, užtikrinant, kad programinė įranga būtų patikimai išleidžiama bet kuriuo metu.

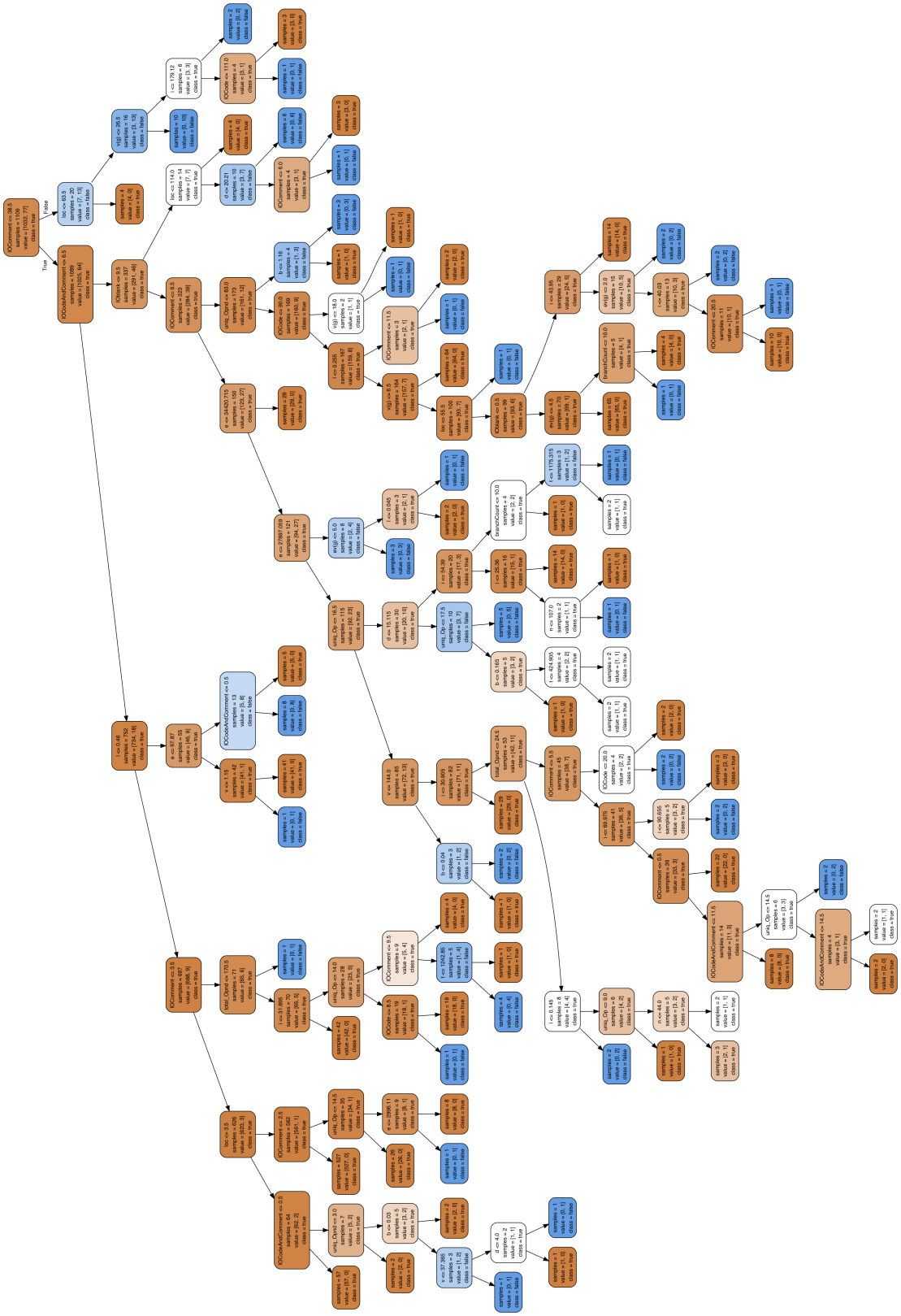
Programos modulis – nepriklausoma programų sistemos dalis.

Operatorius – algoritminės kalbos simbolis, rodantis atliktiną operaciją.

Operandas – matematinės operacijos (funkcijos) argumentas, dydis, su kuriuo atliekamas veiksmas.

Priedas nr. 1

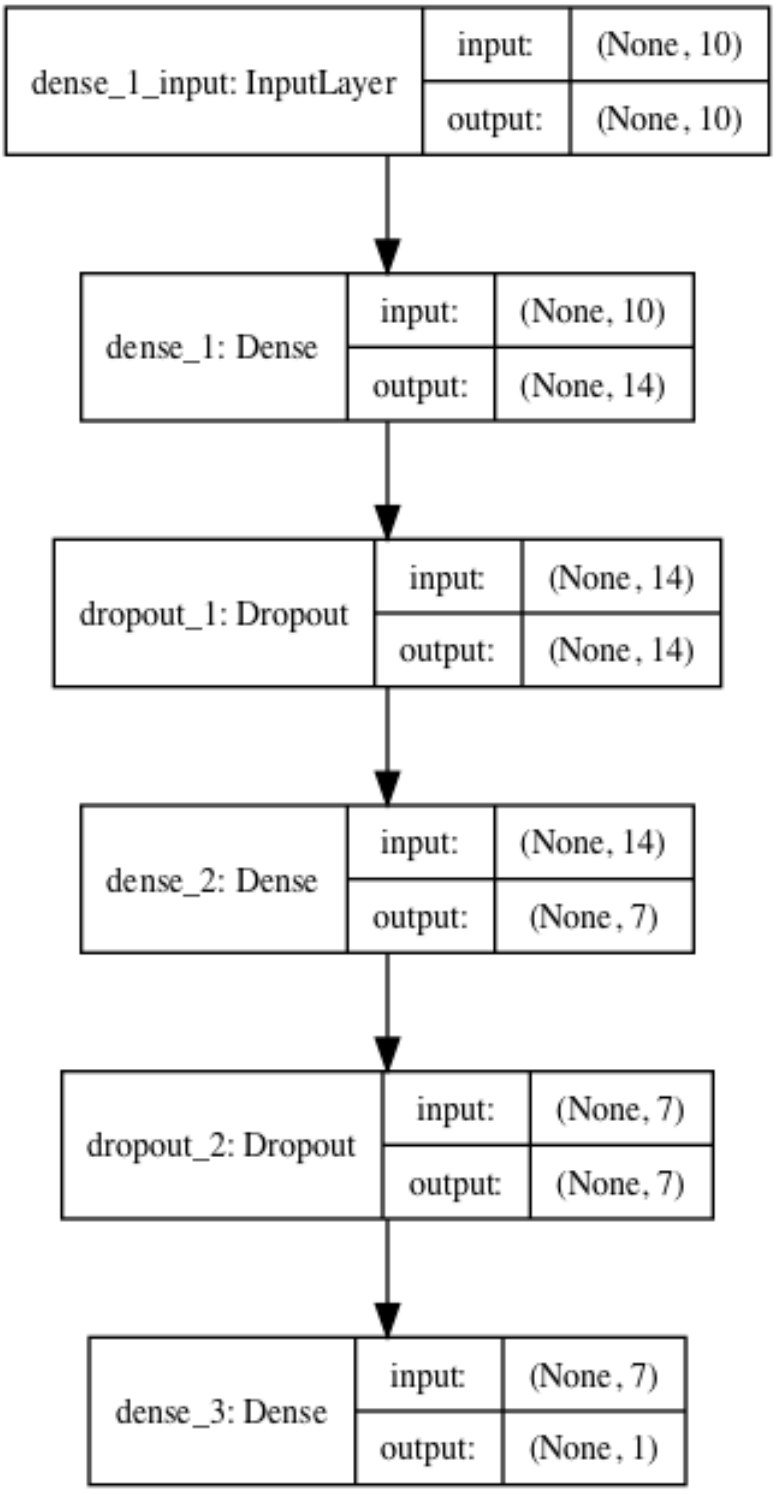
Sprendimų medžio besimokanti sistema



8 pav. Sprendimų medžio tipo besimokanti sistema sudaryta naudojant PC1 duomenų rinkinio poaibį. Sugeneruota automatiškai naudojant pydot biblioteką.

Priedas nr. 2

Atrinktiems rezultatams pritaikyto daugiasluoksnio perceptrono struktūra



9 pav. Daugiasluoksnio perceptrono struktūra atnaujinta atrinktiems atributams. Sugeneruota automatiškai keras.utils.plot_model bibliotekos pagalba.