VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Tiesinis neuroninių tinklų kombinavimas

Linear Combination of Neural Networks

Magistro baigiamasis darbas

Atliko: Mark Juša (parašas)

Darbo vadovas: Arūnas Janeliūnas (parašas)

Recenzentas: Dr. Rimantas Kybartas (parašas)

Vilnius – 2018

Santrauka

Darbe aprašoma į ką reikia atsižvelgti konstruojant sujungtas klasifikatorių sistemas (SKS), kas yra perceptronas ir neuroninis tinklas (NT), kaip sukurti NT. Išanalizuoti kitų autorių darbai kai kombinuojami NT ir kai kombinavimui naudojamas perceptronas. Išanalizuotos įvairios teoriškai optimalių svorių skaičiavimo (TOJS) funkcijos. Atliktas tyrimas. Šio darbo tikslas ištirti ar galima pagerinti SKS darbo kokybę, panaudojant TOJS kombinatoriaus-perceptrono pradiniam svorių inicializavimui. Atliktas tyrimas parodo, kad SKS kokybę pagerinti galime, tačiau atsitiktiniais svoriais inicializuotas kombinatorius-perceptronas dažniau grąžins geresnį rezultatą.

Raktiniai žodžiai: sujungtų klasifikatorių sistemos, teoriškai optimalūs kombinavimo svoriai, perceptronas, dirbtiniai neuroniniai tinklai

Summary

The paper describes what to consider when constructing multi-classifier systems (MCS), what is perceptron, neural network (NN), how to create NN. We analyzed the work of other authors when they combined NN and used perceptron as combination function. Various types of theoretically optimal weights calculation (TOWC) functions are analyzed. Finished experiments. The aim of this work is to investigate whether it is possible to improve the quality of MCS using the initial TOWC as initialization of weights of combiner-perceptron. The carried out research shows that we can improve the quality of MCS, but the combinator-perceptron initialized by random weights will more often return a better result.

Keywords: multi-classifier systems, optimal combination weights, perceptron, artificial neural networks

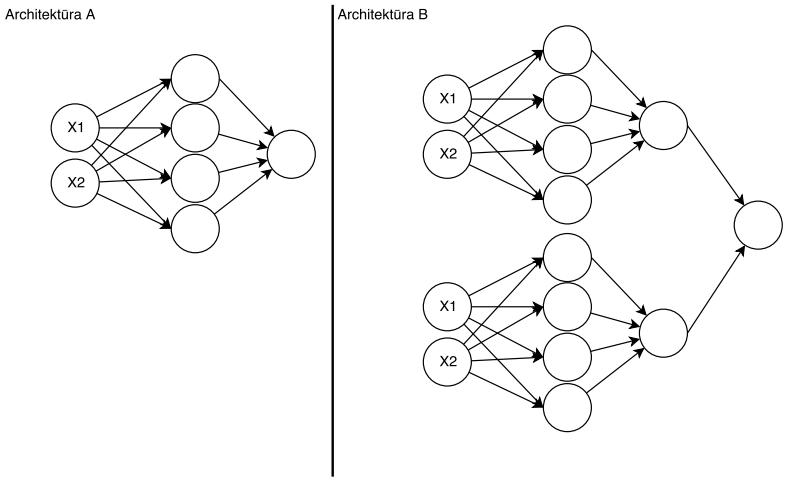
Turinys

1. Įvadas

Jau buvo įrodyta [KV+95; RK96], kad kombinuojant dirbtinius neuroninius tinklus (DNT), mes gausime neblogesnius rezultatus nei naudodami vieną geriausią DNT. Su DNT galime spręsti klasifikavimo uždavinį, todėl DNT galime laikyti klasifikatoriumi. Sujungtų klasifikatorių sistemos (SKS) – sritis, kurioje tiriama kaip geriausiu būdu sujungti kelis klasifikatorius taip, kad jie grąžintų kuo geresnį rezultatą. Buvo pasiūlyta įvairių klasifikatorių kombinavimo būdų. Šiame darbe išanalizuosime DNT kūrimą, perceptrono mokymą, SKS kūrimą ir ištirsime įvairius NT kombinavimo būdus.

DNT galime vadinti matematinį modelį, nusakantį kažkokią nežinomą funkciją [YI08; MSJ+12]. Šis modelis buvo taip pavadintas, nes buvo sukonstruotas biologinio neuroninio tinklo pagrindu. DNT susideda iš neuronų, kurie yra tarpusavyje sujungti. Neuroną laikysime iš anksto nustatyta funkcija, kuri įvesties duomenis suagreguoja ir grąžina vieną skaitinį atsakymą. Toliau DNT vadinsime tiesiog neuroniniu tinklu (NT). NT yra plačiai naudojami klasifikavimo ir regresijos uždaviniams spręsti. Plačiau apie NT konstravimą yra aprašoma 3 skyriuje. Keletas iš šiuo metu tiriamų sričių: bandoma analizuoti žmogaus smegenų bangas ir klasifikuoti jas, bandant sukurti žmogaus-kompiuterio interfeisą pasinaudojant atpažintomis bangų klasėmis [GM88; WDF15], nuotraukų aprašymų generavimas [KF15], kalbos ir kalbėtojo atpažinimas [RRD15] ir kiti panaudojimai.

Iškyla klausimas, kam kombinuoti NT, jeigu galime tiesiog pakeisti NT parametrus, t.y. vietoje to, kad patys kombinuotume gautus NT atsakymus, sudarome praplėstą tinklo architektūrą, kurioje bus kombinuojamas kažkoks skaičius NT.



1 pav. Architektūrų kombinavimo vaizdas

Pvz.: Turime du pasirinktus NT su architektūra 2-4-1 (1 pav. Architektūra A, įėjimo vektoriaus dimensija – paslėpto sluoksnio neuronų skaičius – išėjimo sluoksnio neuronų skaičius). Keleto tokių NT gautus atsakymus galėtume kombinuoti mūsų nustatyta funkcija $y = nt_1 \cdot w_1 + nt_2 \cdot w_2$, kur nt_i yra i-tojo NT atsakymai, o w_i yra tam tikri svoriai. Vietoje kombinavimo pasinaudojant y sudarome NT su architektūra 2-8-2-1 (1 Pav. Architektūra B). Tačiau šitaip praplėsta NT architektūra negali būti prilyginta NT kombinavimui. Kaip žinome, NT mokymo

parametrų parinkimas nėra paprastas uždavinys ir dažnai priklauso nuo duomenų imties [Hay09]. Taip pat nėra aiški nepilnai sujungtų paskutinio sluoksnio neuronų, įtaka NT mokymui. Be to, ne visada yra galimybė praplėsti NT architektūrą, kai jį yra ganėtinai didelė, pvz. dėl atsirandančios dingstančių gradientų problemos, kas lemia ilgesnį NT mokymo laiką (plačiau apie tokių NT mokymą galime paskaityti [Sch15a]). Vis dėlto, SKS kūrimas irgi nėra paprastas uždavinys. Plačiau apie SKS kūrimą yra aprašoma 2.2 skyriuje.

Perceptronas, kaip klasifikatorius, sugeneruoja hiperplokštumą, kuri atskiria vieną klasę nuo kitos, todėl tokį kombinavimo būdą vadinsime tiesiniu. Toliau darbe nagrinėsime būtent šito perceptrono mokymo parametrų parinkimą.

1.1. Darbo aktualumas

Tarkime, kad mes turime kažkokį skaičių NT, kuriuos norėtume pabandyti sujungti ir taip pagerinti klasifikavimo tikslumą. Darbe [Has93] yra aprašyti gana paprastai apskaičiuojami teoriškai optimalūs tiesinio kombinavimo svoriai. Norint apskaičiuoti optimalius kombinavimo svorius, reikėtų žinoti duomenų pasiskirstymo funkciją. Kadangi realioms duomenų imtims mes dažnai nežinome duomenų pasiskirstymo funkcijos, mes galime tik aproksimuoti šituos kombinavimo svorius ir galutiniame variante šitie teoriškai optimalūs kombinavimo svoriai gali būti nevisai optimalūs arba išviso nepaskaičiuojami. Toliau šitame darbe optimalių svorių aproksimavimą laikysime tiesiog optimalių svorių skaičiavimu. Optimalaus kombinavimo uždavinys susiveda į $\vec{w_{opt.}}$ gavimą, kuris galėtų būti apskaičiuotas taip:

$$\vec{w_{opt.}} = \Psi^{-1}\vec{U} \tag{1}$$

Platesnį Ψ ir $\vec{U}$ aprašymą galima rasti 4.1 skyriuje. Šitų kombinavimo svorių optimalumas pasireiškia tuo, kad mes iš statistinių duomenų savybių išskaičiuojame mažiausią galimą klaidą grąžinančius svorius. Svoriai naudojami šitokioje kombinavimo funkcijoje:

$$\sum_{j=1}^{n} w_{opt.j} C_j + w_{opt.0}. \tag{2}$$

Čia C_j – klasifikatoriaus grąžinamas atsakymas, n – klasifikatorių kiekis. Darbe [Has93] yra analizuojama, kaip $\vec{w_{opt.}}$ gali būti panaudojamas regresijos uždaviniams spręsti. Taip pat minima, kad nesudėtingai galėtume pritaikyti $\vec{w_{opt.}}$ klasifikavimo uždaviniui spręsti, tačiau nėra tiksliai apibrėžiama kaip.

Klasifikavimas pasinaudojant perceptronu gali būti aprašytas tokia funkcija:

$$v = \sum_{j=1}^{n} w_j x_j + b, \quad \phi(v) = \begin{cases} 1 & \text{jei } v \geq 0 \\ -1 & \text{jei } v < 0 \end{cases} \tag{3}$$

Platesnis perceptrono aprašymas pateikiamas 3 skyriuje.

Palyginus 2 ir 3 funkcijas galime įžvelgti, kad

$$w_{opt.j} \Longleftrightarrow w_j,$$

$$C_j \Longleftrightarrow x_i,$$

$$w_{opt.0} \Longleftrightarrow b.$$

Atsižvelgiant į tai, $\vec{w_{opt.}}$ galėtume panaudoti inicijuojant perceptrono svorius. Kadangi $\vec{w_{opt.}}$ yra tik aproksimacija, o mokant perceptroną jo svoriai atsinaujina atsižvelgus į daromas klaidas, galimai pavyktų atnaujinti $\vec{w_{opt.}}$ svorius taip, kad bendra klaida dar sumažėtų.

Darbe [Woz07] buvo atliekami eksperimentai kombinuojant penkis klasifikatorius. Taip pat buvo pasiūlytas kitas kombinuojamų klasifikatorių svorių apskaičiavimo algoritmas – adaptuotas svorių skaičiavimo algoritmas (ASS, angl. adaptive weight calculation). ASS buvo išvestas iš perceptrono mokymo algoritmo. ASS gali būti inicijuotas su tam tikrais pradiniais svoriais $\vec{w}$. Darbe svoriai buvo inicijuoti $\frac{1}{k}$, kur k – klasifikatorių skaičius. ASS algoritmo pagalba bandoma išskaičiuoti $\vec{w_{opt.}}$. ASS naudoja statistinius parametrų įvertinimus svorių koregavimui. Atliktas tyrimas parodė, kad apskaičiavus $\vec{w_{opt.}}$ su ASS pagerinsime klasifikatorių kombinavimą. Darbo autorius taip pat nurodo platesnio tyrimo poreikį, kad galima būtų objektyviai spręsti ar ASS verta naudoti klasifikatorių kombinavime.

Atsižvelgus į tai, kad perceptroną galėtume inicijuotas $\vec{w_{opt.}}$, $\vec{w_{opt.}}$ yra aproksimacija, darbe [Woz07] analizuotas ASS algoritmas, kuris yra sukurtas perceptrono mokymo algoritmo pagrindu, sugebėjo atnaujinti kombinavimo svorius, pagerinant klasifikavimo tikslumą kyla klausimai:

1. Kokius rezultatus gausime palyginus NT kombinuojant su 2 funkcija ir kitais NT kombinavimo būdais?

2. Ar dar mokant perceptroną inicijuotą $\vec{w_{opt.}}$ galėtume pagerinti klasifikavimo tikslumą?

Šiai dienai nebuvo rasta tyrimų, kurie ištirtų ar galima panaudoti optimalius kombinavimo svorius kombinuojant NT naudojantis perceptronu [San11; Sch15b; Zho13]. Taip pat buvo atliekama paieška „Google Mokslinčius“[1] paieškos sistemoje.

Šiame darbe mes bandysime pasinaudoti jau žinomais statistiniais parametrų įvertinimais, o NT pranašumas yra tai, kad jis yra neparametrinis, todėl jis gali labiau adaptuotis prie realaus gyvenimo uždavinių. Parametriniams klasifikatoriams gauti rezultatai gali būti sėkmingai pritaikomi tokių NT mokyme.

1.2. Darbo tikslas

Pagrindinis šio darbo tikslas: ištirti ar galima pagerinti SKS darbo kokybę, panaudojant teoriškai optimalius kombinavimo svorius (TOJS) kombinatoriaus-perceptrono pradiniam svorių inicijavimui. Papildomi darbo tikslai: iš įgyvendintų TOJS algoritmų surasti vieną geriausią TOJS

[1] `https://scholar.google.lt/`

skaičiavimo algoritmą, palyginti neapmokomus SKS kombinatorius, apmokomus SKS kombinatorius ir vieną, tiksliausiai klasifikuojantį NT.

1.3. Uždaviniai

Suformuluoti uždaviniai:

1. Klasifikatorių ir jų kombinavimo literatūros analizė.
2. Neuroninių tinklų ir jų kūrimo literatūros analizė.
3. Algoritmų, kurie gali paskaičiuoti optimalius kombinavimo svorius literatūros analizė.
4. Kombinuotų NT ir NT kombinavimo naudojantis perceptronu literatūros analizė.
5. Apmokyti NT.
6. Pritaikyti TOJS klasifikavimui pasinaudojant perceptronu.
7. Pritaikyti TOJS taip, kad galėtume TOJS paskaičiuoti klasifikavimui kai turime daugiau nei dvi klases.
8. Tarpusavyje palyginti apmokomus NT kombinavimo būdus, neapmokomus NT kombinavimo būdus ir vieną tiksliausią NT.
9. Tarpusavyje palyginti kelis TOJS algoritmus.
10. Ištirti ar inicijuotais TOJS perceptronas gali atnaujinti svorius taip, kad klasifikavimo tikslumas padidėtų.

1.4. Laukiami rezultatai

Įvairių NT kombinavimo būdų palyginimas. Įvertinimas, ar TOJS panaudojimas NT kombinuojant perceptronu, patikslina klasifikavimą.

Atsižvelgiant į tai, šį darbą skirstome į tokius skyrius: 2 skyriuje yra aprašoma į ką reikia atsižvelgti konstruojant SKS; 3 skyriuje yra aprašoma kas yra perceptronas ir NT, kaip sukurti NT; 4 skyriuje yra aprašoma kas buvo atlikta kitų autorių darbuose kai yra kombinuojami NT ir kai kombinavimui naudojamas perceptronas, taip pat aprašomi įvairūs TOJS algoritmai; 5 skyriuje yra aprašomas atliktas tyrimas.

Šiame darbe NT jungimo tikslumą apibrėšime santykiu: aibės teisingai klasifikuojamu objektu kiekis su visu aibės klasifikuojamu objektu kiekiu ir žymėsime τ. Pasinaudodami šituo tikslumu, vertinsime NT jungimo būdus tarpusavyje.

Šiame darbe tyrimas buvo atliktas pasinaudojant Tensorflow 1.4 [AAB+15] mašininio mokymo biblioteka. Taip pat buvo naudotos kitos pagalbinės bibliotekos: NumPy, Pandas, Matplotlib, scikit-learn. Plačiau apie panaudotus programinius įrankius yra aprašyta 5 skyriuje.

2. Klasifikatoriai ir jų junginiai

Šiame skyriuje yra aprašomi klasifikatoriai ir jų jungimo būdai. Pagrindiniai šio skyriaus šaltiniai yra [RP06; WGC14]. Šiuose šaltiniuose pateikiami pagrindai SKS kūrimui ir aprašomi to laikotarpio pasiekimai šitoje srityje.

2.1. Klasifikatorius

Klasifikavimas – tai procesas, kurio metu mes tam tikrus objektus skirstome į pastovias klases. Klasifikatoriumi laikysime matematinį modelį, sugebantį pateiktiems duomenims priskirti klasę:

$$F : A_x \mapsto A_y$$

čia A_x – įėjimo vektorių aibė, A_y – išėjimo vektorių aibė, F – matematinis modelis. Procesas, kuriuo metu klasifikatorius yra priverčiamas priskirti tam tikrą objektą į iš anksto žinomą klasę, vadinamas klasifikatoriaus mokymu. Klasifikatoriaus mokyme naudojami objektai sudaro mokymo aibę. Klasifikatorius galime išskirti į dvi kategorijas:

1. Taisyklėmis paremti (angl. rule-based) – tai klasifikatoriai, kuriuose jų kūrėjas nustato pagal kokias taisykles objektai bus klasifikuojami.

2. Skaičiavimais paremti (angl. computational intelligence) – tai klasifikatoriai, kuriuose kūrėjas nustato tik bendrus klasifikatoriaus parametrus, o pačios taisyklės, pagal kurias objektas bus klasifikuojamas, yra išskaičiuojamos algoritmais.

Skaičiavimais paremtus klasifikatoriaus algoritmus galime išskirti į dvi kategorijas:

1. Kontroliuojami (angl. supervised).

2. Nekontroliuojami (angl. unsupervised).

Kontroliuojami algoritmai – tai algoritmai, kurie reikalauja papildomos informacijos apie objektą. Papildomą informaciją apie objektą suteikia mokytojo funkcija. Mokytojo funkcija gali nurodyti tikrąją objekto klasę. Su jos pagalba galime paskaičiuoti skirtumą tarp tikrosios ir algoritmo paskaičiuotos klasės. Turint skirtumus, algoritmas gali pakoreguoti vidinius parametrus atitinkamai tam skirtumui. Nekontroliuojami algoritmai – tai algoritmai, kurie nereikalauja papildomos informacijos apie objektą ir patys bando išskaičiuoti objektų klases. Kontroliuojamo algoritmo pvz.: atgalinio sklidimo algoritmas [Hay09]. Nekontroliuojamo algoritmo pvz.: savi organizuojantys žemėlapiai [Koh90]. Žymėsime: C – klasifikatorius, $A_x = [x_1, x_2, ...]$ – visų galimų įeities vektorių aibė, $A_y = [1,2,...,m]$ – galimų klasių aibė, m – galimų klasių kiekis. Klasifikatoriaus grąžinamus atsakymus galime paskirstyti į tokias kategorijas:

1. Abstraktūs [XKS92] – klasifikatorius grąžina vieną klasę, kuriai priklauso objektas, $C\colon A_x \to A_y$.

2. Ranginiai [XKS92] – klasifikatorius grąžina eilę klasių, kurioms manoma priklauso objektas, surūšiuota pagal tikėtinumą, $C\colon A_x \to B, B = \mathbb{P}(A_y)$.

3. Matuojami [XKS92] – kiekvienai klasei grąžinama tikimybė, kad objektas priklauso tai klasei, $C\colon A_x \to B, B = \mathbb{P}(A_y \times R_{[0,1]}), R_{[0,1]} = \{x\colon \mathbb{R} \mid 0 \leq x \leq 1\}$.

4. Orakulas [Kun14] – klasifikatoriui paduodamas objektas ir jo klasė, o grąžinama ar objektas priklauso pateiktai klasei, $C\colon A \to B, A = A_x \times A_y, B = [0, 1]$.

Darbe [XKS92] išskiriamos trys kategorijos problemų, iškylančių kombinuojant klasifikatorius:

1. I tipo problemos – kai yra jungiami abstrakčius ir orakulo tipo atsakymus grąžinantys klasifikatoriai. Šituo būdu kombinuoti galime bet kokio tipo klasifikatorius.

2. II tipo problemos – kai yra jungiami ranginius atsakymus grąžinantys klasifikatoriai. Iš klasifikatoriaus reikalaujant grąžinti surūšiuotą klasių sąrašą yra apribojamas galimų kombinuojamų klasifikatorių skaičius.

3. III tipo problemos – kai yra jungiami matuojamus atsakymus grąžinantys klasifikatoriai. Labiausiai ribojantys klasifikatorių pasirinkimą, kadangi klasifikatorius turi grąžinti tikimybes (tikėjimą), kad konkretus objektas priklauso klasei.

2.2. Klasifikatorių jungimas

Sugeneravę kelis skirtingus klasifikatorius, gausime kelis matematinius modelius, nusakančius tam tikras nežinomas funkcijas [YI08; MSJ+12]. Buvo įrodyta [Has93; KV+95; RK96], kad jungiant klasifikatorius, mes gausime geresnius rezultatus nei naudodami vieną geriausią klasifikatorių. Kartu teigiama, kad kuo skirtingesni, klasifikatoriai tuo geresnį rezultatą gausime (palyginus su geriausiu klasifikatoriumi). Kriterijus, pagal kurį klasifikatoriai laikomi skirtingais – yra skirtingumas (angl. diversity). Iki šios dienos buvo pasiūlyta daug įvairių skirtingumo matavimo būdų. Vis dėlto, nėra aišku kaip galėtume sukurti SKS, kad gautume geriausią rezultatą. Autorių darbuose [RP06; WGC14] yra išskiriami trys dalykai, į kuriuos reikėtų atsižvelgti konstruojant SKS:

1. Skirtingų klasifikatorių konstravimas.

2. Klasifikatorių topologijos pasirinkimas.

3. Klasifikatorių jungimo funkcijos pasirinkimas.

Sekančiuose poskyriuose plačiau aprašysime kiekvieną iš šitų punktų.

Norėdami gauti gerus rezultatus, turėtume pasirinkti tinkamus klasifikatorius, tačiau nėra aišku kaip apibrėžti klasifikatoriaus tinkamumą. Galėtume paimti ir paskaičiuoti rezultatus su visomis galimomis klasifikatorių kombinacijomis ir taip pasirinkti geriausiai tinkančius. Vis dėlto, Šitoks klasifikatorių pasirinkimas nėra praktiškas, kadangi tokių kombinacijų gali būti labai

daug. [Lam00] minimi trys abstraktūs kriterijai su kuriais galėtume atrinkti klasifikatorius: ortogonalumas, papildomumas ir nepriklausomumas. Ortogonalumas – tai kaip skiriasi klasifikatorių tarpusavio atsakymai. Papildomumas – skirtumai tarp kiekvieno klasifikatoriaus stiprybių ir silpnybių. Nepriklausomumas – kaip vienas klasifikatorius įtakoja kito klasifikatoriaus atsakymus.

2.2.1. Skirtingų klasifikatorių konstravimas

Didelę įtaką sėkmingam klasifikatorių jungimui daro skirtingų klasifikatorių pasirinkimas. Šiame poskyryje pateiksime kaip galime kurti skirtingus klasifikatorius ir kaip galėtume pamatuoti skirtingumą.

2.2.1.1. Skirtingumo užtikrinimas

Sukurtas klasifikatorius turi sugebėti klasifikuoti dar nematytus objektus. Žinant tai, galime sugalvoti keletą būdų, kaip galėtume sukurti skirtingus klasifikatorius pvz.: kuriant klasifikatorius rinktis tik tam tikrus mokymo aibės poaibius, taip užtikrindami, kad vienas klasifikatorius galėtų gerai atskirti vienos klasės objektus nuo visų kitų. Vienas iš pirmųjų darbų, kuriame buvo pristatyti galimi skirtingumo užtikrinimo būdai yra [Die97]. Toliau pateikiami šitie keturi būdai.

1. Mokymo aibės poaibiai – šituo būdu kiekvienas klasifikatorius yra apmokomas su skirtingais mokymo aibės poaibiais ir taip gauname skirtingus klasifikatorius. Šitą metodą geriausia naudoti kai apmokomas klasifikatorius yra nestabilus t.y. kai maži pakeitimai sukelia didelius pokyčius pvz.: apmokant neuroninį tinklą.

   „*Bagging*“ – vienas iš populiarių metodų, veikia taip: imame n atsitiktinių objektų su grąžinimu iš mokymo aibės **M** ir su šitais n objektų apmokome klasifikatorių. Kitas populiarus metodas yra panašus į „*Bagging*“. Vienintelis skirtumas yra tas, kad objektus iš **M** traukiame pagal tikimybinį pasiskirstymą $p_i(x)$. Pagal paskaičiuotą klasifikatoriaus ϵ_i klaidą atnaujiname $p_i(x)$ ir tada vėl traukiame n objektų. Šitas metodas sėkmingai naudojamas „AdaBoost“ algoritme. Kitas galimas būdas yra analogiškas kryžminiam validavimui. Išskaidome **M** į n dalių (gali turėti arba neturėti bendrų elementų) ir su jomis apmokome klasifikatorių.

2. Atributų pakeitimai – šituo būdu klasifikatorius mokome su skirtingai apdorotais atributais. Vienas iš galimų būdų – konstruoti klasifikatorius pasirenkant skirtingus atributų poaibius. Kitas būdas – konstruoti klasifikatorius tam tikru būdu užkoduojant atributus.

3. Klasių poaibiai – šituo būdu klasifikatorius mokome pakeisdami **M** klasių reikšmes. Vienas iš galimų būdų – pasiimti klasių poaibį ir jam priskirti vieną bendrą klasę, ir su ja apmokyti klasifikatorių.

4. Atsitiktinumo panaudojimas – šituo būdu klasifikatorius mokome naudodami atsitiktinumus. Vienas iš galimų būdų – algoritme galime naudoti atsitiktinai inicijuojamas reikšmes pvz.: NT inicijuodami su skirtingais svoriais taip apmokomi skirtingi NT.

5. Algoritmo parametrų keitimas – šituo būdu klasifikatorius mokome naudodami skirtingus mokymo algoritmo parametrus pvz.: NT parametrai tokie kaip: tinklo topologija, neuronų kiekis, mokymo žingsnis.

2.2.1.2. Skirtingumo matavimas

Šiame poskyryje trumpai apžvelgsime, kokie yra skirtingumo matavimo kriterijai.

Kombinuoti panašius rezultatus, grąžinančius klasifikatorius – nėra prasmės, nes toks kombinavimas nesuteiks pranašumo klasifikuojant objektus. Vis dar nėra apibrėžta bendra skirtingumo sąvoka [Zho12][Kun14]. Autoriai savo darbuose siūlo naudoti įvairias skirtingumo metrikas. Dažniausiai autoriai savo darbuose naudoja skirtingumo metrikas klasifikatorių junginio apkirpimui (angl. pruning) [ADF+15] t.y. sugeneruojama daug klasifikatorių ir tada pagal skirtingumo metriką atsirenkami geriausi klasifikatoriai. Darbe [KW03] yra aprašomi dešimt klasifikatorių skirtingumo metrikų. Skirtingumo metrikos dalinasi:

1. Porinės – skirtos matuoti skirtingumą tarp dviejų klasifikatorių.

2. Neporinės – skirtos matuoti skirtingumą tarp trijų ir daugiau klasifikatorių.

Porinės klasifikatorių skirtingumo metrikos: Q statistika, koreliacijos koeficientas, nesutarimo matas, dvigubos klaidos matas. Neporinės klasifikatorių skirtingumo metrikos: entropijos matas, Kohavi-Wolpers (KW) matas, tarpinio susitarimo matas, sudėtingumo matas, apibendrintas skirtingumo matas, sutapimų trikčių skirtingumo matas. Detaliau šitos metrikos yra aprašytos [KW03]

2.2.2. Klasifikatorių junginio topologijos

Tarpusavyje klasifikatoriai gali būti sujungti įvairiais būdais. Sujungus klasifikatorius vienaip ar kitaip gausime skirtingus rezultatus. Darbe [Sha12] tvirtinama, kad klasifikatorių skirtingumas labiau įtakoja SKS rezultatus, nei pati SKS topologija. Kitame darbe [Kun02] tvirtinama, kad pasirinkus tinkamą topologiją galime gauti geresnius SKS rezultatus. SKS topologijos gali būti padalintos į tokias kategorijas:

1. Sąlyginė topologija [Lam00] – yra sudaroma aibė klasifikatorių, tada pasirenkamas pirmas klasifikatorius ir su juo nustatoma klasė. Jeigu pirmas klasifikatorius nesugebėjo priskirti klasės arba klasė buvo priskirta su maža tikimybe, imamas kitas klasifikatorius. Kito klasifikatoriaus pasirinkimas priklauso nuo SKS kūrėjo nustatytos sąlygos. Viena iš sąlygų galėtu būti tokia: visų pirmą, bandome nustatyti klasę pasinaudodami tais klasifikatoriais, kurie sugeba greičiau priskirti klasę objektui, o tada tik klasifikuoti pasinaudojant lėtesniais klasifikatoriais.

2. Hierarchinė (nuosekli, pasirinkimu paremta) topologija [Lam00] – yra sudaroma eilė klasifikatorių, tada objektui iš eilės skaičiuojame galimai priskiriamas klases. Po kiekvieno paskaičiavimo, galimų klasių skaičius mažėja, taip tikslinant objekto klasę su kiekvienu

paskaičiavimu. Dažnai klasifikatoriai surūšiuojami nuo daugiausiai klaidų darančio iki tiksliausio.

3. Hibridinė topologija [Lam00] – yra sudaroma aibė klasifikatorių ir sukonstruojamas kriterijus kuris leis pasirinkti geriausią klasifikatorių objektui. Kriterijus galėtų būti kokia nors atributo reikšmė.

4. Sudėtinė (lygiagreti, jungimu paremta) topologija [Lam00] – yra sudaroma aibė klasifikatorių ir su visais jais klasifikuojamas objektas. Gauti rezultatai sujungiami ir gaunama galutinė objekto klasė. Jungimas galėtų būti atliekamas balsavimo principu.

5. Duomenų priklausomumo topologija [Wan03] – yra sudaroma aibė klasifikatorių taip, kad kiekvienas klasifikatorius atsako už tam tikrą klasę. Kiekvienam klasifikatoriui paskaičiuojama objekto klasė ir pagal objekto atributus jungiama į galutinį atsakymą.

2.2.3. Kombinatoriai

Turint klasifikatorius, reikėtų sujungti jų atsakymus. Klasifikatorių jungimai, kombinatoriai, gali būti padalinti į tokias kategorijas:

1. Ypatybėmis-vektoriais paremtos [XKS92] – pvz.: neuroniniai tinklai, Bajeso, K-NN

2. Sintaksiškai ir struktūriškai paremtos [XKS92] – pvz.: taisyklėmis paremti sprendimai

3. Tiesinis kombinavimas [Pau01] – naudojant tiesines funkcijas pvz.: sumos arba sandaugos funkcijos.

4. Netiesinis kombinavimas [Pau01] – naudojant daugumos balsavimą ir kitas netiesines funkcijas.

5. Statistinis [Pau01] – naudojantis Bajeso kombinavimu arba Dempseter-Shafer technika.

6. Skaičiavimais paremti [Pau01] – naudojant neuroninius tinklus arba genetinius algoritmus.

7. Pasitikėjimu paremti [XKS92] – kiekvienas klasifikatorius turi jam priskirtą svorį, kuris stiprina/slopina klasifikatoriaus atsakymą.

8. Nepasitikėjimu paremti [XKS92] – klasifikatoriai neturi jiems priskirtų svorių

9. Fiksuoti [Dui02] – SKS kūrėjas nustato kombinavimo taisykles

10. Mokomi [Dui02] – Algoritmas nustato kombinavimo taisykles

2.2.3.1. Dažnai sutinkamos kombinavimo funkcijos

Pagal [RP06] dažniausiai sutinkamos kombinavimo funkcijos: čia žymėsime C_i – i-tojo klasifikatoriaus grąžinamas atsakymas, C_{ij} – i-tojo klasifikatoriaus j-tosios klasės priklausymo tikimybė, n – klasifikatorių kiekis, m – klasifikatoriaus galimų klasių kiekis

1. Max – pasirenkama ta klasė, kuriai tam tikras klasifikatorius grąžino didžiausią reikšmę,

$$SKS_{Max} = max\{C_1, C_2, ..., C_n\}.$$

   Šitas jungimas tinka jungti matuojamus rezultatus, grąžinančius klasifikatorius.

2. Min – pasirenkama ta klasė, kuriai tam tikras klasifikatorius grąžino mažiausią reikšmę,

$$SKS_{Min} = min\{C_1, C_2, ..., C_n\}.$$

3. Sum – sumuojami atsakymai gauti iš atskirų klasifikatorių,

$$SKS_{Sum} = \sum_{j=1}^{n} C_j w_j.$$

   Sumavimas be svorių yra atskirtas sumos atvejis kai $\forall i\colon\ w_i = 1$

4. Prod – dauginami atsakymai, gauti iš atskirų klasifikatorių,

$$SKS_{Prod} = \prod_{j=1}^{n} C_j w_j.$$

   Jei norime jungti matuojamus rezultatus grąžinančius klasifikatorius, reikėtų naudoti:

$$SKS_{Prod} = arg \max_{i=1}^{m} \prod_{j=1}^{n} C_{ji}$$

5. Avg – SUM atsakymas dalinamas iš klasifikatoriaus skaičiaus,

$$SKS_{Avg} = \frac{SKS_{Sum}}{n}.$$

6. Avgv – klasifikatoriai veikia balsavimo principu. Kiekvienas klasifikatorius gali atiduoti balsą vienai klasei. Daugiausiai balsų surinkusi klasė išrenkama kaip kombinavimo atsakymas.

$$SKS_{Avgv} = arg \max_{i=1}^{m} \left(\frac{1}{n} \sum_{j=1}^{n} C_{ji} \right)$$

   čia $C_{ij} \in [0,1]$ – i-tojo klasifikatoriaus j-tosios klasės priklausymo balsas.

7. Avgw – toks pats kaip Avgv, tik kiekvieno klasifikatoriaus atsakymas sudauginamas iš tam tikro svorio ir tada balsuojama.

$$SKS_{Avgw} = arg \max_{i=1}^{m} \left(\frac{1}{n} \sum_{j=1}^{n} w_j C_{ji} \right)$$

8. Borda – klasifikatorius grąžina eilę surūšiuotų klasių. Eilės vieta reiškia taškų kiekį priskirtą klasei. Visų klasifikatorių klasių taškai sumuojami ir išrenkama didžiausią taškų kiekį surinkusi klasė.

9. Mv – kiekvienas klasifikatorius grąžina vienintelę klasę. Išrenkama ta klasė, kuri daugiausiai pasikartojo.

10. Bajeso – tiesiškai kombinuojami klasifikatorių tikimybiniai spėjimai pagal jų aposteriorines tikimybes. Čia $p(j|x)$ – aposteriorinės tikimybės, x – objektas, j – klasė

$$p(w_i|x) = \sum_{j=1}^{n} p(w_i|x, C_j)p(C_j)$$

11. Dempster-Shafer – naudojantis įrodymu teorija pasirenkamas atsakymas.

Kitų autorių minimos kombinavimo funkcijos:

1. Tavg [SAA13] – iš AVG išmetame svorius mažesnius už λ, dar vadinamas apkarpytu vidurkiu.

$$SKS_{Tavg} = \frac{\sum\limits_{w_j>\lambda} C_j w_j}{\sum\limits_{w_j>\lambda} 1}$$

2. Median [SAA13] – pasirenkama klasės mediana.

$$SKS_{Median} = mediana\{C_1, C_2, ..., C_n\}.$$

3. ASS [Woz07] – adaptuotas svorių skaičiavimo algoritmas (ASS, angl. adaptive weight calculation), išvestas iš perceptrono mokymo algoritmo.

4. TOJS – naudojant Sum kombinavimo funkciją, kur svoriai w_i yra paskaičiuojami pasinaudojant kažkokiais teoriškai optimaliais jungimo svoriais, plačiau aprašytas 4 skyriuje.

5. Perc – perceptronas plačiau aprašytas 3 skyriuje.

6. Perc+TOJS – perceptronas inicijuotas TOJS, plačiau aprašytas 4 skyriuje.

2.2.3.2. Mokomi kombinatoriai

Mokomi kombinatoriai adaptuojasi pasinaudodami **M**. Pagal [Dui02] kai naudojame tokius kombinatorius, reikėtų atsižvelgti į tokius dalykus:

1. Kiekvieno klasifikatoriaus kalibravimą – kiekvienas klasifikatorius turi būti mokomas atskirai ir jų atsakymai turi būti pakoreguoti vienodam rėžiui.

2. Pasirinkimą ir svorio priskiriamą kiekvienam klasifikatoriui – klasifikatoriai tarpusavyje gali skirtis našumu, kuris gali būti pamatuotas paprastu būdu. Pagal gautą matą, mes pasirenkame klasifikatorius ir priskiriame jiems svorius. Šitokį metodą vadiname globaliu pasirinkimu. **M** gali būti padalinta į kelias dalis ir kiekvienai daliai bus įvertinami klasifikatoriai. Šitoks metodas vadinamas lokaliu pasirinkimu.

3. Globalų kombinavimo metodų pasirinkimą – pasirenkama geriausia kombinavimo strategija.

2.3. Šiame darbe naudojami klasifikatorių kombinavimo metodai

Šio skyriaus pagalba tiksliai aprašėme kokios krypties SKS konstruosime. Taip pat šio skyriaus pagalba nustatėme į ką reikėtų atsižvelgti, kai konstruojame SKS. Atsižvelgiant į tai, kad šiame darbe kaip klasifikatorius naudosime NT, buvo nustatyta:

1. Jungsime kontroliuojamais skaičiavimais paremtus klasifikatorius.

2. Klasifikatoriai grąžins matuojamus atsakymus.

3. Naudojama sudėtinė klasifikatorių topologija.

4. Skirtingumas bus užtikrintas įvairiai inicijuojant NT pradinius parametrus ir naudojant „*Bagging*“ metodą. Svarbu pabrėžti, jog skirtingumas yra svarbus norint konstruoti tiksliau besijungiančius klasifikatorius, tačiau nėra vieningo mato tam skirtingumui nustatyti. Autoriai pateikia daug įvairių matų ir kiekvienu atveju naudojant skirtingumo matą, reikalinga papildoma analizė [Kun16]. Norint labiau susikoncentruoti į TOJS tyrimą, darome prielaidą, kad mūsų daromų veiksmų užtenka skirtingų klasifikatorių konstravimui užtikrinti.

5. Šiame darbe mes kombinatorius grupuosime pagal darbe [Dui02] pristatytas kombinatorių grupes. Fiksuotus kombinatorius vadinsime neapmokomais kombinatoriais. Mokomus kombinatorius vadinsime apmokomais kombinatoriais.

6. Buvo pasirinkta dalis sutinkamų klasifikatorių kombinavimo funkcijų ir jos suskirstytos į apmokomas ir neapmokomas kombinavimo funkcijas. Naudojamos neapmokomos klasifikatorių kombinavimo funkcijos – Max, Min, $Prod$, Avg, $Avgv$, $Best$. Naudojamos apmokomos klasifikatorių kombinavimo funkcijos – $TOJS$, $Perc$, $Perc + TOJS$.

3. Neuroniniai tinklai

Šitame skyriuje yra aprašyta kas yra bendras NT sukūrimas, perceptronas, perceptrono mokymo algoritmas, neuroninio tinklo architektūros ir kaip yra mokomas neuroninis tinklas.

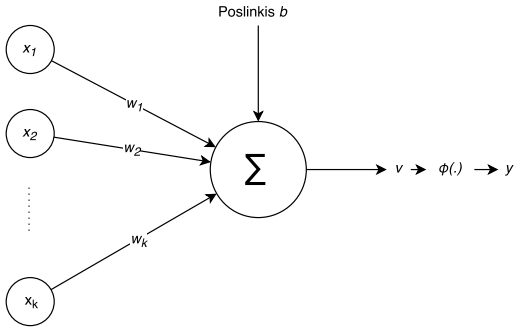
NT konstravimą apibrėšime trejeto parinkimu: tinklo architektūra, neuronai ir mokymo algoritmas. Tinklo architektūra – nusakome, kokiu būdu duomenis bus perduodami neuronams, kiek yra neuronų ir kaip jie yra sujungti. Neuronai – kokio tipo neuronai bus naudojami. Mokymo algoritmas – kokiu metodu nustatomi NT laisvieji parametrai (svoriai ir pan.). Aprašę anksčiau paminėtą trejetą, turėtume sugebėti sukonstruoti NT. Galime tai laikyti kaip NT pradinių parametru parinkimu. Po NT parametrų parinkimo, reikia jį apmokyti. NT apmokymas – tai procesas, kurio metu mes bandome sukonstruotą NT priversti klasifikuoti įvesties objektą į jam iš anksto priskirta klasę. NT apmokymas vykdomas žingsniais, nusakomais pasirinkto mokymo algoritmo. Apmokymo procesą nusako pasirinktas mokymo algoritmas. NT apmokymui yra naudojama mokymo aibė **M**. **M** – tai pradiniai duomenys, su kuriais apmokysime NT klasifikuoti. Šitos aibės vienas objektas susideda iš dviejų vektorių: $\{(x_m, y_m),...\}, m \in \mathbb{N}$ kur x – objektą nusakantys atributai, y – objekto klasė. Sukonstruoto NT vertinimui naudojame testinę aibę **V**. **V** – šitos aibės struktūra tokia pati kaip ir **M**, gali skirtis tik aibėje esantys elementai. Paprastai kalbant, apmokome NT su **M** ir vėliau skaičiuojame tikslumą. Apmokyto NT tikslumą apibrėšime santykiu: testinės aibės teisingai klasifikuojamų objektų kiekis su visu testinės aibės klasifikuojamu objektu kiekiu ir žymėsime τ. NT laikome sukurtu, kai yra pasirinkti jo pradiniai parametrai ir galima pradėti mokymo procesą. NT mokymo proceso metu NT gali patekti į 3 būsenas:

1. NT yra neišmokytas (angl. underfit) – jeigu sekančiuose mokymo žingsniuose τ didėja;
2. NT yra apmokytas (angl. fit)– jeigu sekančiuose mokymo žingsniuose τ nekinta tam tikruose rėžiuose;
3. NT yra permokytas (angl. overfit)– jeigu sekančiuose mokymo žingsniuose τ mažėja;

Toks būsenų kitimas paaiškinamas tuo, kad NT mokyme yra naudojama **M**, o τ yra skaičiuojamas su **V** . Mokymo pradžioje NT netiksliai klasifikuoja tiek **M**, tiek **V** objektus ir NT yra laikomas neišmokytu. Tam tikrame mokymo žingsnyje yra pasiekiama maksimali τ reikšmė ir NT yra laikomas apmokytu. Toliau tęsiant NT mokymą, τ gali pradėti mažėti, dėl to, kad NT prisitaiko prie **M** objektų. Kai tik τ pradeda mažėti NT yra laikomas permokytu. Plačiau apie NT konstravimą ir mokymą galime paskaityti [Hay09].

3.1. Perceptronas

Perceptronas – tai paprasčiausias neuroninis tinklas, naudojamas tiesiškai atskiriamų klasių klasifikavimui. Perceptroną 1958 metais sugalvojo Frankas Rosenblatas (angl. Frank Rosenblatt) [Ros58]. Rosenblato perceptronas susideda iš tiesinio kombinatoriaus ir slenkstinės funkcijos, atvaizduota 2 paveiksle.



2 pav. Perceptrono modelis

Paveiksle su nr. 2, x_i – objekto atributai, w_i – svoriai susieti su perceptronu, $\sum$ – sumatorius, b – poslinkis, v – perceptrono grąžinamas atsakymas, $\phi(.)$ – slenkstinė funkcija, y – grąžintas perceptrono atsakymas, k – objekto atributų skaičius. Matematiškai perceptroną galime aprašyti taip:

$$v = \sum_{i=1}^{k} w_i x_i + b, \quad \phi(v) = \begin{cases} 1 & \text{jei } v \geq 0 \\ -1 & \text{jei } v < 0 \end{cases}$$

Perceptronas mokomas naudojantis tokiu algoritmu:

1. Parametrai ir reikšmės –

$$x(n) = [1, x_1(n), x_2(n), ..., x_k]^T$$

$$w(n) = [1, w_1(n), w_2(n), ..., w_k]^T$$

$$b = \text{poslinkis}$$

$$y(n) = \text{paskaičiuotas atsakymas}$$

$$d(n) = \text{laukiamas atsakymas}$$

$$\gamma = \text{mokymo konstanta nuo 0 iki 1}$$

2. Inicijavimas – nustatome $w(0) = 0$, tada darome sekančius žingsnius kai $n = 1,2,...$

3. Aktyvacija – n-tajame žingsnyje apskaičiuojame x vektoriaus $d(n)$

$$d(n) = \begin{cases} 1 & \text{jei } x(n) \text{ priklauso pirmai klasei} \\ -1 & \text{jei } x(n) \text{ priklauso antrai klasei} \end{cases}$$

4. Atsakymo paskaičiavimas – paskaičiuojame

$$y(n) = \phi(w^T(n) - y(n))$$

5. Svorių pakeitimas – atnaujiname svorius

$$w(n+1) = w(n) + \gamma[d(n) - y(n)]x(n)$$

6. Pratęsimas – eiti į 3 žingsnį, jei n yra nedidesnis už atitinkamą konstantą arba bendra perceptrono klaida nedidėja

Čia aprašytas perceptronas susideda iš vieno neurono. Bendrai neuroną galime išskaidyti į tris dalis:

1. Jungiamieji ryšiai (w_i) – kitaip dar vadinami sinapsėmis. Kiekvienas objekto atributas jungiamas su svorių w_i. Dažnai dar žymimas w_{ji}, kur j – tam tikro neurono indeksas, i – konkretaus svorio indeksas, susietas su atributu. Šitie svoriai gali būti teigiami arba neigiami.

2. Sumatorius ($\sum$)– kombinuoja visų objekto atributų ir svorių reikšmes.

3. Aktyvacijos funkcija (ϕ) – sumatoriaus grąžinto atsakymo apribojimas. Dažnai gražintą atsakymą pakeičia į konkretų tam tikro intervalo skaičių.

Šiame skyriuje buvo aprašytas pats paprasčiausias neuroninis tinklas – perceptronas. Perceptronas gali spręsti tik ganėtinai paprastus klasifikavimo uždavinius, norėdami spręsti kiek sudėtingesnį klasifikavimo uždavinį, turėtume naudoti kitokios architektūros neuroninį tinklą.

3.2. Neuroninio tinklo architektūros

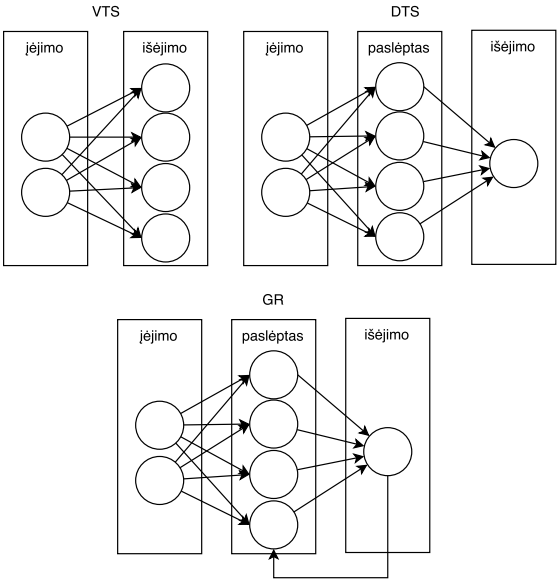
Neuroniai tinklai dalinami į tris pagrindines architektūras [Hay09]:

1. Vienasluoksniai tiesioginio sklidimo (VTS) – šio tinklo neuronai yra organizuojami į įvedimo ir išvedimo sluoksnius.

2. Daugiasluoksniai tiesioginio sklidimo (DTS) – šio tinklo neuronai yra organizuojam į įvedimo, paslėptą ir išvedimo sluoksnius. Palyginus su VTS, šitie tinklai turi paslėptą sluoksnį, kuris padeda iš objekto parametrų ištraukti aukštesnio lygio statistiką. Šitokio tipo tinklai dar gali būti padalinti į pilnai ir dalinai sujungtus. Tinklas yra pilnai sujungtas kai iš kiekvieno sluoksnio išeinančių neuronų atsakymai keliauja į sekančio sluoksnio kiekvieno neurono įėjimus. Tinklai, netenkinantys pilno tinklo reikalavimų, yra daliniai.

3. Grįžtamojo ryšio (GR) – tokiais tinklais vadinami DTS, kuriuose pridedamas grįžtamojo ryšio jungimas t.y. kažkokio neurono grąžinamas atsakymas keliauja ne tik į sekančius sluoksnius, bet ir į prieš tai einančius.

Kiekvienos architektūros pvz.:



3 pav. NT architektūros.

3.3. Neuroninio tinklo mokymas

VTS gali būti apmokomi pasinaudojant perceptrono mokymo algoritmu, tačiau DTS ir GR naudoja kitokius mokymo algoritmus. Tinklo architektūra yra glaudžiai susijusi su mokymo algoritmu. DTS galima apmokyti atgalinio sklidimo (angl. back-propagation) algoritmu arba jo variantais. Panagrinėsime DTS mokymo algoritmo ypatybes. DTS mokymo procesas yra sudėtingas ir dažnai, o norint išmokyti DTS arba pagreitinti jo mokymą, yra naudojamos įvairios euristikos. Toliau pateikiamos kelios tokios euristikos [Hay09]:

1. Stochastinis ir paketinis atnaujinimas – stochastinis atnaujinimas skaičiavimo atžvilgiu yra greitesnis nei paketinis atnaujinimas. Labiausiai tai galima įžvelgti kai mokymo aibė yra didelė ir perteklinė.

2. Maksimaliai padidinti informacijos turinį – kiekvienas pateiktas objektas turi suteikti kuo daugiau informacijos. Kaip pvz.: tai gali būti pasiekta dviem būdais: mokymui naudoti objektus, kurie labiausiai padidina mokymo klaidą arba mokymui naudoti objektus, kurie stipriai skiriasi nuo tų, su kuriais jau NT buvo mokytas.

3. Aktyvacijos funkcija (ϕ) – norint, kad algoritmas mokytųsi greičiau, rekomenduojama naudoti sigmoidinę funkciją, kuri tenkina tokią savybę:

$$\phi(-v) = -\phi(v)$$

4. Tikslo reikšmės ($d(n)$) – svarbu pasirinkti tikslo reikšmes iš sigmoidinės aktyvacijos funkcijos rėžių. Jei pasirinksime neatsižvelgdami į rėžius, NT svoriai gali didėti į begalybę taip sulėtindami mokymą.

5. Objekto atributų normalizavimas – kiekvienas atributas turėtų būti apdorotas taip, kad jo

vidurkis butų artimas nuliui. Taip pat greitį teigiamai įtakoja atributų dekoreliacija ir dekoreliuotų atributų ko-variacijos sulyginimas.

6. Inicijavimas – tinkamai inicijuoti NT svoriai galėtų stipriai pagerinti tinklo mokymąsi. Inicijuoti svoriai turėtų turėti nulinį vidurkį ir dispersiją, susijusią su neuronu jungčių kiekiu.

7. Mokymosi parametras – mokymosi parametras visam tinklui turėtu būti vienodas. Dažnai paskutinio sluoksnio mokymosi parametras yra mažesnis.

3.4. Naudojami NT ir perceptronas

Šio darbo tikslas nėra geriausių parametrų suradimas, todėl užtenka pasirinkti pakankamai gerus rezultatus grąžinančius pradinius NT ir perceptrono parametrus. Tensorflow mašininio mokymo bibliotekoje yra realizuotas DTS ir perceptrono algoritmas. Taip pat Tensorflow bibliotekoje yra pateikiami numatyti DTS ir perceptrono pradiniai parametrai. Kuriant DTS yra siūloma naudoti *Adam* mokymo algoritmą ir *ReLu* aktyvacijos funkciją, kas praktikoje suteikia gana gerus pradinius rezultatus [KB14]. *Adam* algoritmas dinamiškai nustato mokymosi parametrus, taip sumažindamas parametrų parinkimo skaičių. Kuriant perceptroną yra siūloma naudoti *Ftrl* mokymo algoritmą [MHS+13]. Naudojamas *Ftrl* algoritmas reikalauja papildomų parametrų parinkimo, dėl to mes perceptrono mokyme naudosime *Adam* mokymo algoritmą. Perceptronas naudoja tokią aktyvacijos funkciją:

$$h_\theta(x) = \frac{1}{1 + e^{-x}}$$

Toliau šiame darbe DTS ir NT laikysime sinonimais. NT kūrimo metu parinksime greičiausiai besimokančią NT architektūrą ir paanalizuosime mokymo žingsnių kiekį.

Šiame magistriniame darbe didžiausias dėmesys skiriamas perceptrono inicijavimui, kuri bandysime optimizuoti naudodami jau žinomus (optimalius) rezultatus parametriniams klasifikatoriams.

4. Neuroninių tinklų jungimas ir jungimas naudojantis perceptronu

Galima sudaryti įvairius tiesinius ir netiesinius jungimo būdus. Iš vienos pusės, lygtais, NT kombinavimui geriau naudoti netiesines funkcijas, tačiau naudojant tokias funkcijas, junginys gali būti persimokęs. Mes nagrinėsime tiesinius jungimo būdus dėl to, kad su jais galime gauti labiau bendrus duomenis. Šitame skyriuje yra aprašomas NT jungimas ir jungimas naudojantis perceptronu – kokie buvo atlikti tyrimai šitoje srityje ir kokie rezultatai buvo gauti.

4.1. Neuroninių tinklų jungimas

Darbe [RP05] buvo jungiami keturi neuroniniai tinklai. Šie keturi tinklai buvo jungiami trimis būdais: MV, SUM ir LOP2. LOP2 – tai darbe pasiūlytas LOP algoritmo pakeista versija. LOP (angl. logarithmic opinion pool) buvo pristatytas darbe [HK00]. Darbe gauti rezultatai parodo, kad geriausias būdas jungti NT yra pasinaudojus LOP2. Autoriai taip pat mini, kad geresnio rezultato gavimą labiausiai įtakojo didesnės architektūros NT ir skirtingas kiekvieno NT įvesties parametrų kodavimas. Verta paminėti, kad sudėtingiausiu aspektu autoriai laiko NT tinkamų parametrų parinkimą.

Darbe [SB00] pateikiamas „AdaBoost“ algoritmas pritaikytas neuroniniams tinklams. Buvo tiriami trys skirtingi NT generavimo būdai. Jungimas buvo atliekamas naudojant netiesinę funkciją. NT jungimas buvo palygintas su kitais klasifikatorių jungimais ir parodė akivaizdų pranašumą. Buvo pastebėta, kad naudojant AdaBoost su NT, junginys nepersimoko.

Darbe [GVC05] buvo nagrinėjami skirtingų NT jungimo būdų našumas. Buvo pristatytas SimAnn algoritmas, kuris suranda optimalų NT junginį ir jį kombinuoja pasinaudojant AVG. Paprastai sakant, optimalus junginys randamas generuojant daug NT ir pasirenkant geriausiai tarpusavyje susijungiančius NT. Taip pat buvo naudojamas kitas NT junginio generavimo algoritmas SECA, kuriame sugeneruotu NT kombinatoriumi buvo AVG. Palyginus su kitais algoritmais, SimAnn ir SECA parodė geriausius rezultatus. Verta paminėti, kad algoritmai buvo naudojami regresijos uždaviniui spręsti.

Darbe [SAA13] pasiūlyti GA-NNE ir SA-NNE algoritmai su kuriais galima būtų surasti optimalų NT junginį. GA-NNE – tai genetiniu algoritmu paremto optimalaus NT junginio paieškos algoritmas. SA-NNE – tai simuliuoto atleidimo (angl. sumulated annealing) algoritmu paremtas optimalaus NT junginio paieškos algoritmas. Šie algoritmai remiasi principu, kad sukuriama daug NT ir tada algoritmo pagalba surandamas geriausias junginys. Šitie metodai buvo palyginti su GASEN algoritmu, kuris yra aprašytas žemiau, ir parodė geresnius rezultatus. Šitų algoritmų pagalba pasirenkami tiek NT, tiek jų jungimo būdai. Naudoti jungimo būdai: AVG, SUM, MEDIAN, TAVG.

4.2. Optimalių tiesinio kombinavimo svorių radimo būdai

Kaip jau minėjome 1 skyriuje, siekiant apskaičiuoti optimalius svorius, galime pasinaudoti:

$$\vec{w_{opt.}} = \Psi^{-1}\vec{U}.$$

Norėdami paskaičiuoti $\vec{w_{opt.}}$, turime: $\Psi = [\psi_{ij}] = [\mathbf{E}(y_i(\vec{X})y_j(\vec{X}))]$ yra $(N+1)\times(N+1)$ matrica, o $\vec{U} = [u_i] = [\mathbf{E}(r(\vec{X})y_i(\vec{X}))]$ yra $(N+1)\times 1$ vektorius, N – NT kiekis. Darbe minima, kad šios funkcijos, optimalius svorius turėtų skaičiuoti gausiniams duomenims. Praktikoje mes dažniausiai nežinome duomenų pasiskirstymo funkcijos, todėl turime aproksimuoti $\Psi^{-1}\vec{U}$. Tolimesniuose poskyriuose pateikiame darbe [Has93] pateikiamas TOJS aproksimacijų funkcijas.

4.2.1. TOJS

Darbe išskiriama kaip pagrindinė TOJS skaičiavimo funkcija. TOJS skaičiuojame panaudodami N NT.

$$\widehat{\Psi} = [\psi_{ij}] = \left[\sum_{k=1}^{m} y_i(\vec{X}_k)y_j(\vec{X}_k))/m\right]$$

$$\widehat{\vec{U}} = [u_i] = \left[\sum_{k=1}^{m} r(\vec{X}_k)y_i(\vec{X}_k))/m\right]$$

čia $y_j(\vec{X})$ – NT atsakymai, $r(\vec{X})$ – tikrasis atsakymas, $y_0(\vec{X}) = 1$, m – duomenų objektų kiekis, $\vec{w_{opt.}}$ – $(N+1)\times 1$ vektorius.

4.2.2. TOJS-A

Kitaip apskaičiuojama TOJS funkcija. Skaičiuojant šitą funkciją, vieno NT atsakymas panaudojamas kaip apribojimas. Toliau remiantis šia funkcija apskaičiuotus TOJS vadinsime TOJS apribotais (TOJS-A).

$$\widehat{\Psi^{-1}} = [\psi_{ij}] = \left[\sum_{k=1}^{m}(y_i(\vec{X}_k) - y_c(\vec{X}_k))(y_j(\vec{X}_k) - y_c(\vec{X}_k)))/m\right]$$

$$\widehat{\vec{U}} = [u_i] = \left[\sum_{k=1}^{m}(r(\vec{X}_k) - y_c(\vec{X}))(y_i(\vec{X}_k) - y_c(\vec{x}))/m\right]$$

čia $c \in [1,2,...,m], c \neq j$, $y_0(\vec{X}) = a, a \in \mathbb{R}$ (darbe [Has93] $a = 1$), $\vec{w_{opt.}}$ – $N \times 1$ vektorius.

4.2.3. TOJS-BP

Kitaip apskaičiuojama TOJS funkcija. Skaičiuojant šitą funkciją yra nenaudojamas poslinkis. Toliau remiantis šia funkcija apskaičiuotus TOJS vadinsime TOJS be poslinkio (TOJS-BP).

$$\widehat{\Psi} = [\psi_{ij}] = \left[\sum_{k=1}^{m} y_i(\vec{X}_k)y_j(\vec{X}_k))/m\right], (i,j > 0)$$

$$\widehat{\vec{U}} = [u_i] = \left[\sum_{k=1}^{m} r(\vec{X}_k) y_i(\vec{X}_k))/m \right], (i > 0)$$

čia $w_{opt.}^{\rightarrow}$ – $N \times 1$ vektorius.

4.2.4. TOJS-ABP

Kitaip apskaičiuojama TOJS funkcija. Skaičiuojant šitą funkciją, vieno NT atsakymas panaudojamas kaip apribojimas ir nenaudojamas poslinkis. Toliau remiantis šia funkcija apskaičiuotus TOJS vadinsime TOJS apribotais be poslinkio (TOJS-ABP).

$$\widehat{\Psi^{-1}} = [\psi_{ij}] = \left[\sum_{k=1}^{m} (y_i(\vec{X}_k) - y_c(\vec{X}_k))(y_j(\vec{X}_k) - y_c(\vec{X}_k)))/m \right], (i,j > 0)$$

$$\widehat{\vec{U}} = [u_i] = \left[\sum_{k=1}^{m} (r(\vec{X}_k) - y_c(\vec{X}))(y_i(\vec{X}_k) - y_c(\vec{x}))/m \right], (i > 0)$$

čia $w_{opt.}^{\rightarrow}$ – $(N-1) \times 1$ vektorius.

4.2.5. ATOJS-A

Kitaip apskaičiuojama TOJS-A funkcija. Toliau remiantis šia funkcija apskaičiuotus TOJS vadinsime alternatyviais TOJS apribotais (ATOJS-ABP).

$$w_{opt.}^{\rightarrow} = \frac{\widehat{\Omega}^{-1}\vec{1}_z}{\vec{1}_z^t \widehat{\Omega}^{-1}\vec{1}_z}$$

$$\widehat{\Omega} = [\omega_{ij}] = \left[\sum_{k=1}^{m} (\delta_i(\vec{X}_k)\delta_j(\vec{X}_k))/m \right]$$

čia $\delta_i(\vec{X}_k) = r_i(\vec{X}_k) - y_i(\vec{X}_k)$, $\vec{1}_z = (0,1,1.....)$, $w_{opt.}^{\rightarrow}$ – $(N+1) \times 1$ vektorius.

4.2.6. ATOJS-ABP

Kitaip apskaičiuojama TOJS-A funkcija. Skaičiuojant šitą funkciją, nenaudojamas poslinkis. Toliau su šia funkcija apskaičiuotus TOJS vadinsime alternatyviais TOJS apribotais be poslinkio (ATOJS-ABP).

$$w_{opt.}^{\rightarrow} = (0, w_{opt.}^{\prime\rightarrow})$$

$$w_{opt.}^{\prime\rightarrow} = \frac{\widehat{\Omega}^{-1}\vec{1}}{\vec{1}^t \widehat{\Omega}^{-1}\vec{1}}$$

$$\widehat{\Omega} = [\omega_{ij}] = \left[\sum_{k=1}^{m} (\delta_i(\vec{X}_k)\delta_j(\vec{X}_k))/m \right]$$

čia $\delta_i(\vec{X}_k) = r_i(\vec{X}_k) - y_i(\vec{X}_k)$, $\vec{1} = (1,1,1.....)$, $w_{opt.}^{\rightarrow}$ – $N \times 1$ vektorius.

Darbe taip pat minima, kad norint gauti optimalų $w_{opt}^{\rightarrow}$ kombinuojant NT reikėtų juos parinkti taip, kad jie būtų nekolinearūs. Šitie optimalūs kombinavimo svoriai buvo taikomi regresijos

uždaviniui spręsti.

4.2.7. GASEN

Darbe [ZWT02] pateikiama funkcija

$$\vec{w_{opt.}}_k = \frac{\sum_{j=1}^{N} C_{kj}^{-1}}{\sum_{i=1}^{N}\sum_{j=1}^{N} C_{ij}^{-1}}$$

su kuria galime apskaičiuoti optimalų svorių vektorių $\vec{w_{opt}} = (\vec{w_{opt.}}_1, \vec{w_{opt.}}_2, ..., \vec{w_{opt.}}_N)$. Čia C – koreliacijos matrica, N – NT kiekis. Galime bandyti C išskaičiuoti iš turimų duomenų X:

$$C_{ij} \approx \frac{1}{N}\sum_{n=1}^{N}[(y_n - f_i(x_n))(y_n - f_j(x_n))]$$

Praktikoje jos taikyti negalime dėl sekančių priežasčių: gauname panašius (klasifikavimo prasme) NT ir dėl to negalime apskaičiuoti C^{-1} (matrica yra išsigimusi arba negalime tiksliai paskaičiuoti jos atvirkštinės)[ZWT02]. $\vec{w_{opt}}$ skaičiavimą galime apibrėžti kaip optimizavimo problemą. Genetiniai algoritmai gana gerai sprendžia optimizavimo uždavinius, todėl [ZWT02] sukonstruotas GASEN algoritmas paremtas genetiniais algoritmais su kurio pagalba bandome aproksimuoti $\vec{w_{opt}}$ ir nuspręsti kokius NT reikėtų palikti, o kokius išmesti. Kaip teigiama darbe [ZWT02], jeigu naudojame GASEN algoritmą ir skaičiuojame $\vec{w_{opt}}$ kuri naudosime NT kombinavime, klasifikuojant SKS lengvai persimokys, todėl NT kombinavime naudojamas MV. Kai GASEN naudojame regresijos uždaviniui spręsti, SKS nepersimoko. Toliau pateikiamas GASEN algoritmas:

1. Naudodami mokymo aibę X, apmokome T neuroninių tinklų N_i kiekvieną kartą iš X pasirinkdami objektus su grąžinimu.
2. Sugeneruojame atsitiktinius svorius w ir nustatome svorių ribą λ.
3. Naudojantis standartiniu genetiniu algoritmu, pristatytu [Gol89], evoliucionuojame svorius; Naudojame pritaikymo funkciją $f(w) = 1/E_w^V$, čia E_w^V – klasifikatorių junginio klaida paskaičiuota naudojantis svoriu w ir validavimo aibe V.
4. Genetinis algoritmas grąžina $\vec{w_{opt.}}$.
5. Jei sprendžiame regresijos uždavinį, jungiame klasifikatorius:

$$N^*(x) = \frac{\sum\limits_{w_{opt.i}>\lambda} N_i}{\sum\limits_{w_{opt.i}>\lambda} 1}.$$

6. Jei sprendžiame klasifikavimo uždavinį, jungiame klasifikatorius:

$$N^*(x) = \underset{y\in Y}{argmax} \sum_{w_{opt.i}>\lambda : N_i(x)=(y)} 1.$$

4.2.8. TTRVM-DTS

[CTY09] svarstomas kiek kitoks $\vec{w_{opt}}$ skaičiavimas. [CTY09] laikoma, kad tai yra išskaidyta Bajeso mokymo problema (angl. sparse Bayesian learning problem), sprendžiama Tippingo tiesioginio ryšio vektorių mašina (TTRVM, angl. Tipping's relevance vector machine), suformuluota [Tip01]. Viena problema su TTRVM, kad skaičiuojant $w_{opt.k}$ jie gali būti neigiami, o tai neigiamai įtakoja SKS [BSE+97][Ued00]. Darbe [CTY09] bandoma išvengti TTRVM problemų panaudojant deterministinį tikėtiną sklidimą (DTS, angl. expectation propagation) aprašytą [Min01]. Algoritmo trumpas aprašymas:

1. Sugeneruojame N klasifikatorių ir inicijuojame pradinius atvirkštinės dispersijos svorius α.

2. Mokome DTS algoritmą su svoriais α ir iš eilės atnaujiname vektoriaus α elementus maksimizuodami antro tipo ribinio tikėtinumo (angl. type-II marginal likelihood) funkciją $p(X|\alpha)$. Priklausomai nuo to kaip keitėsi α nusprendžiame ar pridėti naują klasifikatorių į junginį, ar išmesti klasifikatorių iš junginio, ar perskaičiuoti α. Kartojame šį žingsnį kol algoritmas nesukonverguos. Algoritmui sukonvergavus, grąžinamas aproksimuotas optimalus svorių vektorius $w_{opt.}$.

3. Pasirenkame NT junginį su mažiausia klaida. Vykdant šį algoritmą, klaidos yra apskaičiuojamos su kiekviena iteracija, todėl galime pasirinkti junginį su mažiausia klaida.

4.3. Optimalių tiesinio kombinavimo svorių panaudojimas perceptrono inicijavime

Realizuosime 4 pagrindinius darbe [Has93] pristatytus svorius: TOJS, TOJS-A, TOJS-BP, TOJS-ABP. Darbe [Has93] TOJS laikoma kaip pagrindinė svorių apskaičiavimo funkcija, nes ja remiantis ir apskaičiuojame teoriškai optimaliai jungiančius svorius. Kitos funkcijos negarantuoja teorinio optimalumo.

Perceptrono inicijavime naudosime tik TOJS. Pagrindinė to priežastis: dalis funkcijų nenaudoja poslinkio (TOJS-BP, TOJS-ABP), o kita dalis naudoja vieno NT atsakymą kaip poslinkį, kas sudaro keblumu apibrėžiant slenkstį tarp dviejų klasių.

TOJS buvo skirtas regresijos uždaviniui spręsti. Šiame darbe sprendžiame klasifikavimo uždavinį. Todėl reikia paanalizuoti TOJS ir pritaikyti juos klasifikavimo uždaviniui.

4.3.1. TOJS kai duomenų imtis gali būti klasifikuojama į 2 klases

Kai imties objektą galime klasifikuoti tik į 2 klases, iš esmės mums užtenka 1 atsakymo, ką ir grąžina perceptronas. Šiame darbe apmokomi NT grąžina matuojamus atsakymus. Kai NT suklasifikuoja objektą, yra grąžinamos tikimybės, kad imties objektas priklauso 1 ir 2 klasei. Iš esmės galime naudoti tik tikimybę, kad objektas priklauso 2 klasei, nes tikimybė, kad objektas priklauso 1 klasei bus $P(pirma) = 1 - P(antra)$. Taip pat galime apibendrinti tikimybes ir kaip NT atsakymus naudoti klasės numerį. Tikimybių neapibendrinsime, nes toks apibendrinimas iš

esmės suapvalina atsakymą ir taip mes prarandame dalį informacijos. Iš NT atsakymų naudodami antros klasės tikimybes, galime paskaičiuoti TOJS, TOJS-A, TOJS-BP, TOJS-ABP.

$Perc$ yra naudojama logistinė aktyvacijos funkcija, todėl skaičiuojant TOJS reikia pakoreguoti $r(\vec{X})$. Nepakoregavus $r(\vec{X})$, paskaičiuoti TOJS bus netikslūs, kai juos panaudosime perceptrono inicijavime. Taip pat pakoreguosime $y_j(\vec{X})$. Atnaujinti kintamieji apskaičiuojami taip:

$$\widehat{y_j}(\vec{X}) = y_j(\vec{X}) - 0.5,$$

$$\widehat{r}(\vec{X}) = r(\vec{X}) - 0.5.$$

$\widehat{\Psi}$ ir $\widehat{\vec{U}}$ įgauna tokį pavidalą:

$$\widehat{\Psi} = [\psi_{ij}] = \left[\sum_{k=1}^{m} \widehat{y_i}(\vec{X_k})\widehat{y_j}(\vec{X_k}))/m \right]$$

$$\widehat{\vec{U}} = [u_i] = \left[\sum_{k=1}^{m} \widehat{r}(\vec{X_k})\widehat{y_i}(\vec{X_k}))/m \right]$$

Po šitokio atnaujinimo NT galime jungti perceptronu inicijuotu TOJS ir jau pirmame žingsnyje gauti tokį patį tikslumą kaip ir kombinuojant su TOJS.

Perceptronas, kuris kombinuos NT atsakymus:

$$v = \sum_{i=1}^{k} w_i \widehat{y_i} + b, \quad \phi(v) = \frac{1}{1 + e^{-v}}$$

w_i – i-tasis perceptrono svoris, $\phi(v)$ – tikimybė, kad kombinuojamas rezultatas priklauso 2 klasei.

4.3.2. TOJS kai duomenų imtis gali būti klasifikuojama į daugiau nei 2 klases

Kai imties objektą galime klasifikuoti į daugiau nei 2 klases, iš esmės mes negalime panaudoti vieno perceptrono kaip kombinatoriaus, nes jis grąžina tik 1 skaitinį atsakymą. Dėl to $Perc$ buvo panaudoti n perceptronų. n – imties klasių skaičius. Šiame darbe apmokomi NT grąžina matuojamus atsakymus. Kai NT suklasifikuoja objektą, yra grąžinamos tikimybes, kad imties objektas priklauso n-tai klasei. Galime apibendrinti tikimybes ir kaip NT atsakymus naudoti klasės numerį. Šiuo atveju apibendrinsime tikimybes, nors ir prarasime dalį informacijos. Neapibendrinant tikimybių gausime, kad kiekvienas NT grąžina po n tikimybių, kurias turėsime panaudoti kaip perceptrono įeitį. Tokio uždavinio sprendimas tampa nebepanašus į tipinio perceptrono panaudojimą.

$Perc$ yra naudojama logistinė aktyvacijos funkcija, todėl skaičiuojant TOJS reikia pakoreguoti $r(\vec{X})$. Nepakoregavus $r(\vec{X})$, negalėsime naudoti $Perc$, kurio naudojimą apibrėžia Tensorflow. Atnaujinti kintamieji apskaičiuojami taip:

$$\widehat{r}(\vec{X_k}) = (x_i : x_i := 0, i \neq r(\vec{X_k}); x_i := 1, i = r(\vec{X_k}))$$

$\widehat{\vec{U}}$ įgauna tokį pavidalą:

$$\widehat{\vec{U}} = [u_{ij}] = \left[\sum_{k=1}^{m} (\widehat{r}(\vec{X}_k))_j y_i(\vec{X}_k))/m \right]$$

$\vec{w_{opt.}}$ įgauna tokį pavidalą:

$$\vec{w_{opt.}} = \Psi^{-1} \widehat{\vec{U}}$$

čia Ψ yra $(N+1) \times (N+1)$ matrica, $\widehat{\vec{U}}$ yra $(N+1) \times n$ matrica, $\vec{w_{opt.}}$ yra $(N+1) \times n$ matrica.

Po šitokio atnaujinimo, NT galime jungti perceptronu, inicijuotu TOJS ir jau pirmame žingsnyje gauti tokį patį tikslumą kaip ir kombinuojant su TOJS. Perceptronas, kuris kombinuos NT atsakymus:

$$v = \vec{X}_k \vec{w_{opt.}}, \quad \phi(v) = \frac{1}{1+e^{-v}}$$

čia v yra n ilgio vektorius, k yra tam tikro imties objekto indeksas.

Šiame skyriuje buvo apžvelgti įvairūs NT kombinavimo būdai ir optimalių svorių radimo būdai. Apžvelgtuose darbuose buvo daromos skirtingos prielaidos apie duomenis ir dėl to buvo naudojami skirtingi kombinatoriai. Praktikoje optimalius svorius skaičiuojame iš mokymo duomenų, tai tie paskaičiuoti optimalūs svoriai gali būti prisitaikę prie mokymo duomenų. Pasinaudoję perceptronu kaip kombinatoriumi pabandysime inicijuoti jo svorius iš tam tikro paskaičiuoto $w_{opt.}$ ir jį dar pamokyti. Galbūt pamokius tokį perceptroną su mokymo duomenimis pagerės jo klasifikavimo kokybė. Šitokio mokymo tikslas yra nukreiptas į tai, kad perceptronas labiau prisitaiko prie realių duomenų, todėl tikėtina, kad inicijavus jį su jau optimaliais svoriais ir dar pamokius, gausime geresnius rezultatus grąžinantį junginį. Nėra aišku, kada toks perceptronas bus pranašesnis už kitus jungimo metodus.

5. Tyrimas ir jo eiga

Šiame skyriuje yra detaliai aprašomas atliktas tyrimas. 5.1 poskyryje aprašomi kokie įrankiai buvo naudoti ir kas buvo suprogramuota. 5.2 poskyryje aprašome kokias duomenų imtis naudojame. 5.3 poskyryje trumpai aprašome kaip buvo atliktas kiekvienas iš bandymų. 5.4 poskyryje detaliai aprašome atliktus bandymus.

5.1. Naudoti įrankiai

Šis tyrimas buvo atliktas pasinaudojant Tensorflow 1.4 [AAB+15] mašininio mokymo biblioteka. Ji buvo pasirinkta, dėl šių priežasčių: ją galima naudoti su jau žinoma programavimo aplinka Python; žemas mokymosi lygis norint pradėti naudotis aukšto lygio API sąsaja; yra įgyvendintas NT algoritmas su galimybe pasirinkti įvairius mokymo būdus; skaičiavimams atlikti galime naudoti tiek centrinį kompiuterio procesorių (angl. trumpinys CPU), tiek grafikos koprocesorių (angl. trumpinys GPU). Taip pat buvo naudotos kitos pagalbinės bibliotekos: NumPy, Pandas, Matplotlib, scikit-learn. Duomenims saugoti buvo pasinaudota SQLite duomenų baze.

5.1.1. Įgyvendintas funkcionalumas

NT buvo kuriami ir mokomi pasinaudojant biblioteka Tensorflow. Parsisiųsti duomenys buvo transformuoti į bibliotekos palaikomą formatą. Šioje bibliotekoje, šio darbo rašymo metu, nebuvo realizuotos funkcijos, leidžiančios jungti jau sukurtus NT, todėl buvo įgyvendintos tokios jungimo funkcijos:

1. Max, Min, $Prod$, Avg, $Avgv$, $Perc$, $Perc + TOJS$ – jungimo funkcijos aprašytos 2.2.3 poskyryje.

2. $TOJSG$, $NPerc$, $NPerc + TOJS$ – jungimo funkcijos aprašytos šiame skyriuje.

$TOJSG$ – tai NT jungimas pasinaudojant Sum kurio w_i yra paskaičiuoti pagal 4.2 skyriuje aprašytas formules. $TOJSG$ susideda iš keturių dalių: TOJS, TOJS-A, TOJS-BP, TOJS-ABP. $NPerc$ – tai NT jungimas pasinaudojant perceptronu, kurio algoritmas įgyvendintas remiantis [Hay09]. $NPerc$ algoritmas buvo suprogramuotas todėl, kad $Perc$ suteikia ribotas galimybes valdyti vidinius parametrus ir mokymo procesą. $NPerc$ pritaikytas TOJS svorių inicijacijai. $NPerc+TOJS$ – tai NT jungimas pasinaudojant $NPerc$ tik pradiniai jungimo svoriai buvo inicijuoti TOJS gautais svoriais. Biblioteka buvo sukonfigūruota naudojant grafikos koprocesorių.

5.2. Naudoti duomenys

Norint išsamiau išnagrinėti NT jungimą, bus naudojamos 5 sugeneruotos ir 5 realios duomenų imtys. Sintetines imtis naudosime kaip bazę, nes apie šitų duomenų savybes mes žinome daugiau, o papildomai tikrinsime kokius rezultatus NT junginys grąžina su realiomis duomenų imtimis. Sintetinės duomenų imtys buvo sugeneruotos pasinaudojant scikit-learn biblioteka. Paveiksle 4 pavaizduojamos žemiau aprašytos sintetinės imtys. Naudosime skirtingus imties dydžius.

Kiekvienos imties požymius standartizuosime (išcentruota pagal vidurkį su vienetine dispersija). Toliau pateikiami kiekvienos sintetinės imties detalesni aprašymai:

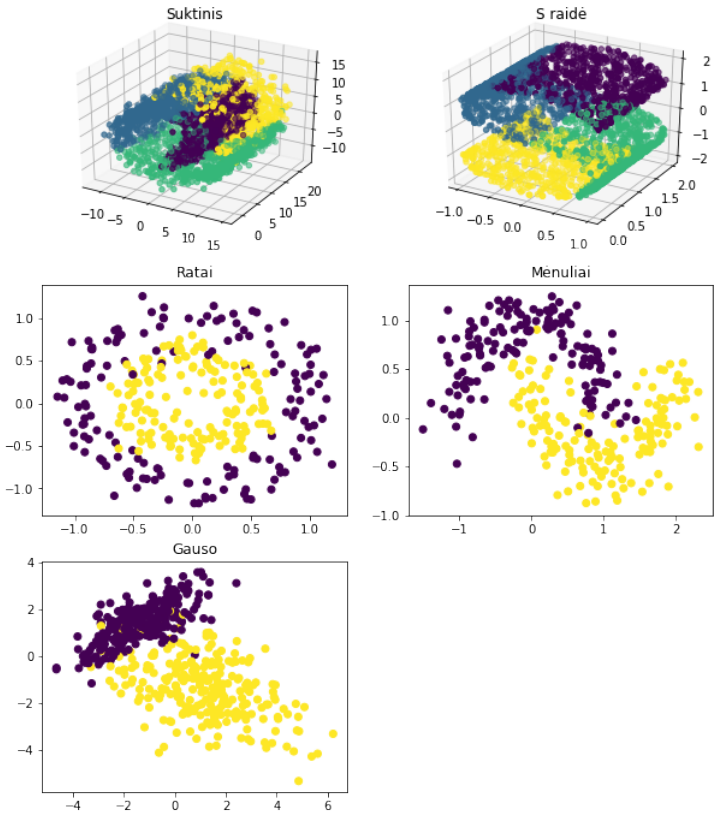
1. Suktinis – šveicariško vyniotinio formos duomenys. Imtis turi 3 požymius ir 4 klases. Klasės tikimybė 0.25. Šitie duomenys generuojami regresijos uždaviniui spręsti, todėl buvo sukurtos 4 sintetinės klasės. Ignoruojant gylio požymį, galime šitą imtį atvaizduoti 2 dimensijų plokštumoje ir gausime spiralę, kurią galėtume nesudėtingai vizualiai atskirti.

2. Ratai – aplink 2 apskritimus einantys taškai ir vienas apskritimas gaubia kitą apskritimą. Imtis turi 2 požymius ir 2 klases. Klasės tikimybė 0.5.

3. Mėnuliai – mėnulio formos duomenys. Imtis turi 2 požymius ir 2 klases. Klasės tikimybė 0.5.

4. S raidė – s raidės formos duomenys. Imtis turi 3 požymius ir 4 klases. Šitie duomenis generuojami regresijos uždaviniui spręsti, todėl buvo sukurtos 4 sintetinės klasės.

5. Gausiniai duomenys – dviejų klasterių formos tipo duomenys, sugeneruoti klasifikavimui. Klasei priklausantys vektoriai erdvėje paskirstyti pagal Gauso skirstinį. Imtis turi 2 požymius ir 2 klases. Klasės tikimybė 0.5.



4 pav. Sintetinių duomenų vaizdas

Naudojamos realios duomenų imtys buvo paimtos iš UCI duomenų talpyklos[Lic13]. Renkantis duomenų imtis, buvo stengiamasi pasirinkti tokias imtis, kuriose dauguma požymių būtų

skaitiniai. Buvo pasirinktos kelios iš populiariausių duomenų imčių. Toliau pateikiami duomenų imčių aprašymai:

1. Iris – šios imties vektoriai nusako iriso augalo tipus. Imtis turi 3 požymius ir 3 klases. Klasė – augalo tipas. Imtį sudaro 150 vektorių po 50 kiekvienai klasei.

2. Vynas – šios imties vektoriai nusako iš skirtingos veislės vynuogių pagamintą vyną. Imtis turi 13 požymių ir 3 klases. Klasė – vynuogės veislės tipas. Imtį sudaro 178 vektoriai, 59 pirmos klasės, 71 antros klasės, 48 trečios klasės.

3. Vyno kokybė – šios imties vektoriai nusako skirtingų vynų kokybę. Imtis turi 11 požymių ir 11 klasių. Klasė – vyno kokybė nuo 0 iki 10. Imtį sudaro 1599 vektorių. Šios imties vektoriaus klasės yra pasiskirsčiusios neproporcingai. [CCA+09]

4. Vyno kokybė – šios imties vektoriai nusako skirtingų vynų kokybę. Imtis turi 11 požymių ir 11 klasių. Klasė – vyno kokybė nuo 0 iki 10. Imtį sudaro 4898 vektorių. Šios imties vektoriaus klasės yra pasiskirsčiusios neproporcingai. [CCA+09]

5. Vėžys – šios imties vektoriai nusako vėžio trijų tipų ląstelių branduolių savybes, nustatytas iš nuotraukų. Imtis turi 30 požymių ir 2 klases. Klasė – ar vėžys piktybinis, ar ne. Imtį sudaro 569 vektoriai, 357 pirmos klasės ir 212 kitos klasės.

5.3. Tyrimo planas

Pagrindinis šio darbo tikslas yra išsiaiškinti ar pasinaudojus teoriškai optimaliais NT jungimo svoriais (TOJS), galima būtų pagerinti NT jungimą pasinaudojant perceptronu. Papildomai atlikti tyrimus su kitais NT jungimo būdais ir juos palyginti. Sintetiniams duomenims TOJS galime apskaičiuoti gana tiksliai, kadangi duomenys buvo sugeneruoti iš tiksliai apibrėžtų funkcijų. Realiems duomenims TOJS galime bandyti skaičiuoti, tačiau tikėtina, jog tie apskaičiuoti svoriai nebus optimalūs, nes duomenys gali turėti didelį triukšmą arba nevienareikšmiškus vektorius. Bendrai, galima pasakyti taip: tikėtina, kad realiems duomenims apskaičiuoti TOJS, NT jungs tam tikrame lokaliame minimume, o inicijavus perceptroną TOJS ir jį pamokius, mes galėtume gauti globalesnį minimumą taip sumažindami NT jungimo klaidą.

Trumpai aprašome kas buvo atlikta bandymuose:

1. *Bandymas* 1 – paimtos visos sintetinės ir nesintetinės duomenų imtys ir su jomis apmokyti NT. Pasirinkta viena greičiausiai besimokanti NT architektūra. Visi su ta NT architektūra apmokyti NT, buvo surūšiuoti pagal **V** tikslumą ir pasirinkta po 10 geriausiai, vidutiniškai ir blogiausiai apmokytų NT. Pasirinkti NT buvo jungiami įgyvendintais, neapmokomais ir apmokomais NT jungimo būdais.

2. *Bandymas* 2 – Atsitiktinai pasirenkame duomenų imtį, kuri gali būti suklasifikuota į 2 klases, NT kiekį ir pačius NT, ir tada šitą pasirinkimą jungiame neapmokomais ir apmokomais NT jungimo būdais. Pasirenkame tik tuos NT, kurie buvo inicijuoti su 1 bandyme

nustatyta greičiausiai besimokančia NT architektūra. Vykdome atsitiktinį pasirinkimą 100 kartų.

3. $Bandymas\,3$ – Atsitiktinai pasirenkame duomenų imtį, kuri gali būti suklasifikuota į 2 klases, NT kiekį ir pačius NT, ir tada šitą pasirinkimą jungiame neapmokomais ir apmokomais NT jungimo būdais. Ribojame NT pasirinkimą pagal **V** tikslumą τ, imame NT, kurie tenkina sąlygą $0.5 > \tau > 0.8$. Vykdome atsitiktinį pasirinkimą 100 kartų.

4. $Bandymas\,4$ – Atsitiktinai pasirenkame duomenų imtį, kuri gali būti suklasifikuota į daugiau nei 2 klases, NT kiekį ir pačius NT, ir tada šitą pasirinkimą jungiame neapmokomais ir apmokomais NT jungimo būdais. Ribojame NT pasirinkimą pagal **V** tikslumą τ, imame tik tuos NT, kurie tenkina sąlygą: $0.5 > \tau > 0.8$. Vykdome atsitiktinį pasirinkimą 100 kartų.

Detalūs kiekvieno bandymo aprašymai pateikti žemiau esančiuose poskyriuose.

5.4. Gauti rezultatai

5.4.1. Bandymas 1

Kadangi NT jungimas priklauso nuo daug faktorių, o kitų autorių gauti atsakymai vienareikšmiškai neišskiria vieno geriausio jungimo būdo, palyginsime dalį jungimo būdų tarpusavyje.

Tikslas: Sukonstruoti bandymams reikalingus NT ir palyginti įvairius NT jungimo būdus.

Numanomas rezultatas: Apmokyti NT. Tarp neapmokomų jungimo būdų neturėtų būti vieno geriausio pasirinkimo.

Aprašymas ir gautų rezultatų analizė:

Kad NT susijungtų sėkmingai, reikia juos kurti skirtingus. Norėdami užtikrinti NT skirtingumą: skirtingai inicijavome NT pradinius svorius ir papildomai naudojome „*Bagging*“ algoritmą. Kadangi NT apsimokymo greitis priklauso nuo imties duomenų ir jų kiekio, buvo bandoma surasti greičiausiai apsimokančią NT architektūrą visoms pasirinktoms duomenų imtims. Naudotos NT architektūros: $X-10-Y$, $X-10-10-Y$, $X-5-5-5-Y$, $X-15-10-5-Y$, $X-15-15-15-Y$, $X-15-15-15-15-Y$, čia X – duomenų požymių skaičius, o Y – duomenų klasių skaičius. NT apmokymui, duomenų imtys buvo padalintos į dvi dalis: **M** ir **V**. **M** sudaro 80% visos duomenų imties. Po **M** sudarymo klasių pasiskirstymas atitiko visos imties klasių pasiskirstymą su 25% paklaida. Mokant kiekvieną NT, jo mokymo aibė **M'** buvo generuojama iš **M**, pasinaudojant „*Bagging*“ algoritmu. **M'** dydis yra lygus **M** , o tai reiškia, kad kiekvienas NT buvo apmokytas su apie 60% **M** duomenų[APR07]. NT buvo mokomi stabdant po tiek žingsnių: $1, 2, 4, 6, 8, 12, 16, 24, 32, 48, 64$. Norint gauti vidutinę pasirinktos architektūros ir žingsnio klaidą, buvo apmokoma 30 NT ir jų tikslumas suvidurkinamas. NT buvo mokomi pasinaudojant *tensorflow.estimator.DNNClassifier* klase. Kaip teigiama *tensorflow* dokumentacijoje, kuriant anksčiau paminėtą klasę su numatytomis reikšmėmis naudojamas *Adam* mokymo algoritmas ir *ReLu* aktyvacijos funkcija, kas praktikoje suteikia gana gerus pradinius rezultatus [KB14].

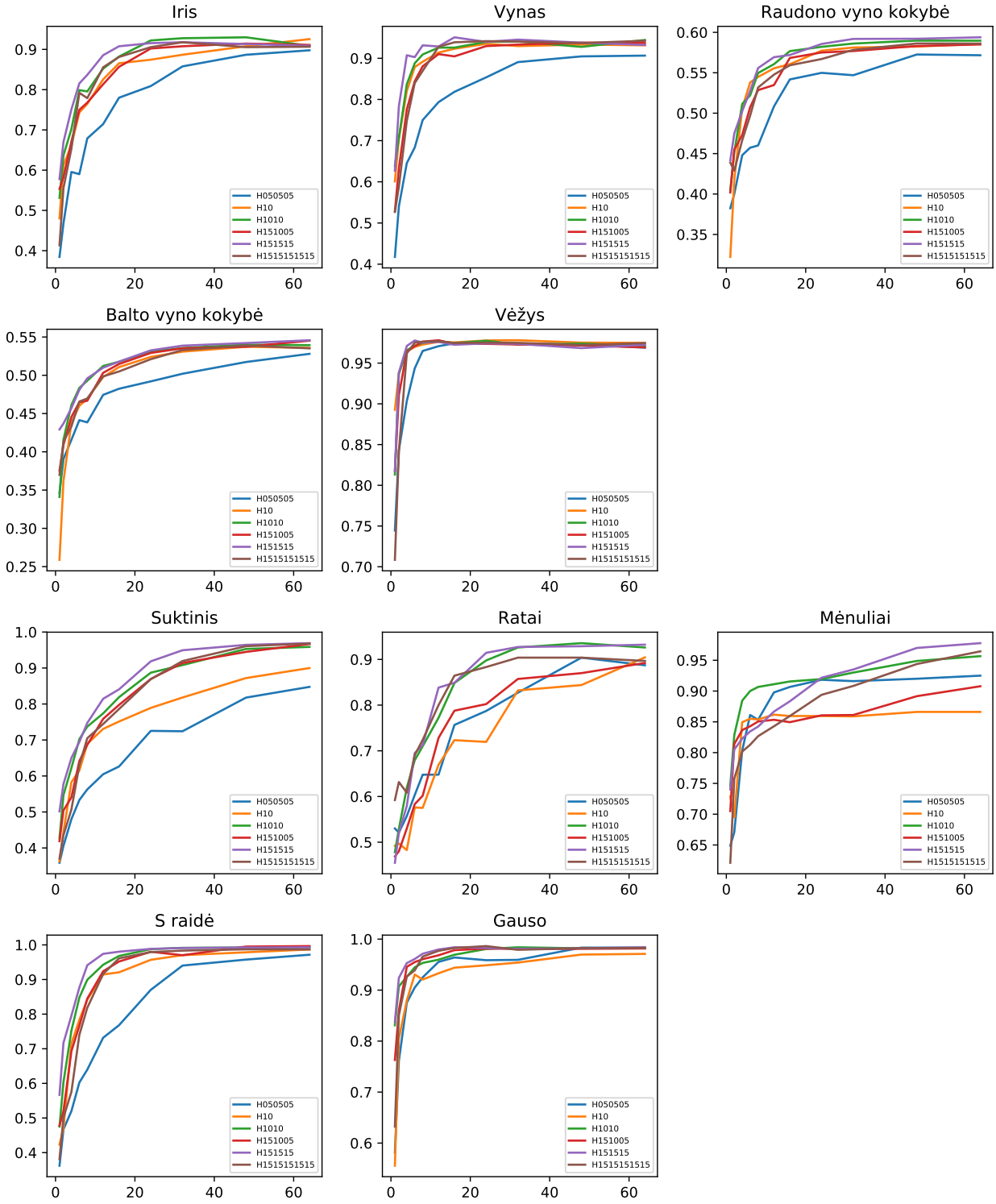
Lentelėje 1 pateikiama apmokytų NT statistika. Aibė – tai su kokia aibe buvo gauti rezultatai. Šitoje lentelėje skaitinės reikšmės – tai teisingai suklasifikuotų vektorių kiekio santykis su visų vektorių kiekiu iš atitinkamos aibės. Atitinkamai: Min – mažiausia, Max – didžiausia, Mean – vidutinė reikšmė, Median – mediana.

1 lentelė. Apmokytu NT rezultatai

Duomenys	Aibė	Min	Max	Mean	Median
Suktinis	**M'**	0.0783	0.9837	0.7192	0.7368
Ratai	**M'**	0.1791	0.9875	0.7727	0.8125
Mėnuliai	**M'**	0.1916	1.0000	0.8581	0.8750
S raidė	**M'**	0.1133	1.0000	0.8210	0.9254
Gauso	**M'**	0.0550	0.9850	0.9125	0.9475
Iris	**M'**	0.0250	1.0000	0.8398	0.8833
Vynas	**M'**	0.1690	1.0000	0.9120	0.9929
Raudono vyno kokybė	**M'**	0.0938	0.7193	0.5534	0.5801
Balto vyno kokybė	**M'**	0.0801	0.5990	0.4848	0.5002
Vėžys	**M'**	0.3560	1.0000	0.9569	0.9846
Suktinis	**V**	0.0833	0.9816	0.7143	0.7333
Ratai	**V**	0.1666	0.9833	0.7236	0.7500
Mėnuliai	**V**	0.1833	1.0000	0.8549	0.8666
S raidė	**V**	0.1033	1.0000	0.8184	0.9233
Gauso	**V**	0.0200	0.9900	0.9270	0.9600
Iris	**V**	0.0333	1.0000	0.7786	0.8333
Vynas	**V**	0.1666	1.0000	0.8402	0.9166
Raudono vyno kokybė	**V**	0.0781	0.6312	0.5260	0.5531
Balto vyno kokybė	**V**	0.0704	0.5683	0.4790	0.4989
Vėžys	**V**	0.3947	0.9912	0.9491	0.9736

Blogiausiai NT apsimokė su vyno kokybės imtimis. Šių imčių klasės yra sudarytos iš žmonių pateiktų vertinimų, todėl galimai šių imčių artimų klasių vektoriai yra panašūs ir tai trukdo NT apmokymui. Jeigu vertinsime likusias imtis, tai geriausias NT šias imtis suklasifikuos su $>= 0.98$ tikimybe.

Apmokius visus NT, buvo bandoma juos jungti įvairiais būdais. Paveiksle 5 pavaizduojame kiekvienai imčiai, kiekvienai apmokytai NT architektūrai vidutinę klaidą, mokant NT tam tikru žingsniu.



5 pav. Apmokytų NT klaidų vidurkiai, paskaičiuoti pasinaudojant **V**. Kiekvienas grafikas nusako tam tikrą duomenų imtį, x ašyje mokymo žingsnių kiekis, y ašyje vidutinis tikslumas, tiesė nusako tam tikros architektūros NT

Kaip galime pastebėti paveiksle 5, beveik visoms imtims greičiausiai apsimokė $X-15-15-15-Y$ architektūros NT, todėl jį ir naudosime tolimesnėje analizėje.

Lentelėje 2 pateikiami įvairių NT jungimo būdų rezultatai panaudojus **V** . Kiekvienai imčiai apmokyti NT buvo surūšiuoti pagal **V** klaidą ir pasirinkta po 10 geriausiai, vidutiniškai ir blogiausiai apmokytų NT. Panašiais NT laikysime tokius NT, kurie klasifikavimo metu grąžina tuos

pačius atsakymus arba kitaip sakant klysta dėl tų pačių vektorių. Šitaip padalindami apmokytus NT, tikimės paanalizuoti tokius NT junginius: geriausiai apmokytiems NT – tikėtina tokie NT bus panašūs, vidutiniškai apmokytiems NT – tikėtina tokie NT bus labiau skirtingi, blogiausiai apmokytiems NT – tikėtina tokie NT bus skirtingi. Stulpelio pavadinimas atitinka 2 skyriuje aprašytus jungimo būdus. *Best* – tai geriausio klasifikatoriaus rezultatas.

2 lentelė. NT jungimai

Duomenys	*Max*	*Min*	*Prod*	*Avg*	*Avgv*	*Best*	Apmokyti
Suktinis	0.9733	0.9717	0.9750	0.9750	0.9717	**0.9800**	Geriausiai
Ratai	0.9667	0.9667	0.9667	0.9667	0.9667	**0.9833**	Geriausiai
Mėnuliai	0.9667	0.9667	0.9833	0.9833	0.9833	**1.0000**	Geriausiai
S raidė	0.9917	0.9883	0.9950	0.9950	0.9950	**0.9983**	Geriausiai
Gauso	**0.9900**	**0.9900**	**0.9900**	**0.9900**	**0.9900**	**0.9900**	Geriausiai
Iris	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	Geriausiai
Vynas	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	Geriausiai
Raudono vyno kokybė	0.6062	0.6000	0.6312	0.6281	**0.6344**	0.6313	Geriausiai
Balto vyno kokybė	**0.5714**	0.5622	0.5673	0.5684	0.5684	0.5684	Geriausiai
Vėžys	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	Geriausiai
Suktinis	0.8417	0.8433	0.8450	**0.8483**	0.8367	0.8267	Vidutiniškai
Ratai	0.9500	0.9500	**0.9667**	**0.9667**	**0.9667**	0.9000	Vidutiniškai
Mėnuliai	0.9333	0.9333	0.9333	0.9333	**0.9500**	0.8667	Vidutiniškai
S raidė	**0.9883**	0.9817	0.9817	0.9800	0.9800	0.9767	Vidutiniškai
Gauso	**0.9900**	**0.9900**	**0.9900**	0.9800	0.9800	0.9800	Vidutiniškai
Iris	**0.9333**	**0.9333**	**0.9333**	**0.9333**	0.9000	0.8667	Vidutiniškai
Vynas	0.9722	0.9722	**1.0000**	**1.0000**	**1.0000**	0.9167	Vidutiniškai
Raudono vyno kokybė	0.6000	0.5844	0.5938	0.5844	**0.6031**	0.5781	Vidutiniškai
Balto vyno kokybė	**0.5520**	0.5469	0.5469	0.5480	0.5490	0.5347	Vidutiniškai
Vėžys	0.9649	0.9649	0.9649	**0.9737**	**0.9737**	**0.9737**	Vidutiniškai
Suktinis	0.6967	0.6600	0.7217	**0.7367**	0.6067	0.5433	Blogiausiai
Ratai	**0.5833**	**0.5833**	0.5667	0.5667	0.5667	0.5500	Blogiausiai
Mėnuliai	**0.7667**	**0.7667**	**0.7667**	**0.7667**	0.7167	0.7500	Blogiausiai
S raidė	0.5983	**0.8350**	0.8333	0.8183	0.7583	0.5850	Blogiausiai
Gauso	**0.9300**	**0.9300**	**0.9300**	**0.9300**	0.8800	0.7900	Blogiausiai
Iris	0.5667	**0.6333**	0.5667	0.5667	0.5667	0.5667	Blogiausiai
Vynas	**0.7222**	0.6667	0.6944	0.6944	0.6389	0.6111	Blogiausiai
Raudono vyno kokybė	0.5719	**0.5812**	0.5656	0.5688	0.5656	0.5094	Blogiausiai
Balto vyno kokybė	0.5398	0.5429	**0.5500**	0.5439	0.5480	0.5051	Blogiausiai
Vėžys	**0.9211**	**0.9211**	0.8070	0.7456	0.6228	0.7281	Blogiausiai

Paanalizuokime kiekvieną sudarytą grupę.

Geriausiai klasifikuojantys NT – kaip matome, dažniausiai reikia imti vieną tiksliausią NT, nes 8 iš 10 kartų taip gausime geriausius rezultatus. 4 imtims nebuvo skirtumo kokį jungimo būdą rinktis, o tai reiškia, kad dauguma jungiamu NT, grąžina tuos pačius rezultatus. Su vyno kokybe susijusios duomenų imtys išsiskyrė nuo kitų imčių tuo, kad kombinuojant geriausius NT, gavome geresnius rezultatus nei geriausias NT. Kombinavimo rezultatas buvo 1% geresnis nei geriausias klasifikatorius.

Vidutiniškai klasifikuojantys NT – kaip matome, visada kombinuojant tokius NT gaunama neblogesnius rezultatus nei geriausias NT. 5 kartus iš 10 kombinuojant Avg, buvo neblogesnis nei kiti kombinavimo būdai. Kombinuojant NT, klasifikavimas pagerėja nuo 1% iki 10% palyginus su geriausiu klasifikatoriumi.

Blogiausiai klasifikuojantys NT – kaip matome, visada kombinuojant tokius NT, gaunama geresnius rezultatus nei geriausias NT, 7 kartus iš 10 kombinuojant Min, buvo neblogesnis nei kiti kombinavimo būdai. Kombinuojant NT, klasifikavimas pagerėja nuo 2% iki 43% palyginus su geriausiu klasifikatoriumi. Neskaitant vyno kokybės duomenų imčių, joks NT jungimas neaplenkė bendro geriausio NT klasifikavimo tikslumo. Toliau palyginsime apsimokančius NT kombinavimo būdus.

TOJS galime paskaičiuoti tik tada, kai sprendžiame regresijos uždavinį, o mūsų nagrinėjami NT jungimo būdai yra pritaikyti klasifikavimui. Vis dėlto, 2 klasių klasifikavimą galime nesudėtingai pertvarkyti į regresijos uždavinį, nes turime kiekvienos klasės tikimybes. Tai padarysime taip: pirmai klasei priskirsime -0.5 reikšmę, o antrai klasei 0.5 reikšmę, NT klasifikavimo rezultatas bus tikimybė, kad vektorius priklauso antrai klasei. Iš paskaičiuotos tikimybės atimame 0.5. Šito užtenka norint paskaičiuoti TOJS. Toliau analizuosime tik tas aibes, kurios turi 2 klases. Lentelėje 3 pateikiame NT jungimo rezultatus panaudojus **V** . Šioje lentelėje pateikiami rezultatai apmokomiems NT jungimo būdams. TOJS priskiriame prie apmokomų NT jungimo būdų, nes jungimo svorius turime išskaičiuoti iš **V** arba **M** . Plačiau apie apmokomus NT jungimo būdus: $Perc$ – perceptronas buvo mokomas 5 skirtingais mokymo žingsniais, kiekvienam mokymo žingsniui buvo generuojama 10 perceptronų, jie apmokomi 300 žingsnių ir pasirenkamas geriausiai apmokytas perceptronas. $Perc + TOJS$ – kadangi numanome, kad inicializavus perceptroną optimaliais jungimo svoriais ir jį pamokius galėtume pagerinti NT jungimą, perceptroną mokėme dar 100 žingsnių. $NPerc$ – perceptronas buvo mokomas 7 skirtingais mokymo žingsniais, kiekvienam mokymo žingsniui buvo generuojama 10 perceptronų, jie apmokomi iki tol, kol nepradeda kristi jų tikslumas ir tada pasirenkamas geriausiai apmokytas perceptronas. $NPerc + TOJS$ – perceptronas inicializuotas TOJS buvo mokomas 7 skirtingais mokymo žingsniais iki tol, kol nepradėdavo kristi tikslumas ir tada pasirenkamas geriausiai apmokytas perceptronas. Visi apmokyti perceptronai buvo pasirinkti pagal **M** tikslumą ir tik po pasirinkimo buvo fiksuojamas **V** tikslumas ir jis naudojamas lentelėje. Patogumo dėlei, šioje lentelėje taip pat pateikiame geriausio NT rezultatą.

3 lentelė. NT jungimai

Duomenys	*Best*	TOJS	TOJS-A	TOJS-BP	TOJS-ABP	*Perc*	$Perc + TOJS$	*NPerc*	$NPerc + TOJS$	Apmokyti
Ratai	**0.9833**	0.9000	0.9500	0.9000	0.9333	0.9500	0.9333	0.9500	0.9000	Geriausiai
Mėnuliai	**1.0000**	0.9667	0.9667	0.9667	0.9667	0.9500	0.9500	0.9667	0.9667	Geriausiai
Gauso	**0.9900**	**0.9900**	**0.9900**	**0.9900**	**0.9900**	**0.9900**	**0.9900**	0.9800	**0.9900**	Geriausiai
Vėžys	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	**0.9912**	Geriausiai
Ratai	0.9000	0.8667	0.8667	0.8667	0.8667	0.9333	0.9167	**0.9667**	0.8833	Vidutiniškai
Mėnuliai	0.8667	0.9167	0.8833	0.9167	0.8833	**0.9333**	0.8833	0.9167	**0.9333**	Vidutiniškai
Gauso	0.9800	0.9800	0.9800	0.9800	0.9800	**0.9900**	**0.9900**	**0.9900**	0.9800	Vidutiniškai
Vėžys	0.9737	0.9649	0.9737	0.9649	0.9737	**0.9912**	0.9649	**0.9912**	0.9649	Vidutiniškai
Ratai	0.5500	0.8833	0.6167	0.9167	0.6500	**0.9333**	0.8333	**0.9333**	0.9167	Blogiausiai
Mėnuliai	0.7500	0.8333	0.8500	**0.8833**	0.7667	**0.8833**	0.8333	**0.8833**	**0.8833**	Blogiausiai
Gauso	0.7900	**0.9900**	0.8000	0.9800	0.8000	0.9800	0.9500	0.9800	**0.9900**	Blogiausiai
Vėžys	0.7281	0.9649	0.6053	0.9561	0.9561	**0.9737**	0.9474	**0.9737**	0.9649	Blogiausiai

Paanalizuokime kiekvieną sudarytą aibę.

Geriausiai klasifikuojantys NT – kaip ir su neapmokomais NT jungimo būdais, geriausiai yra naudoti vieną geriausią klasifikatorių. Jungiant tokius NT, $TOJSG$ turėtų apskaičiuoti optimalius svorius, tačiau šituo atveju tie svoriai nebuvo optimaliausi. Tokius rezultatus galėjo įtakoti visų NT panašumas. Darbe [Has93] paminėta, kad jeigu NT bus panašūs ir bandysime skaičiuoti $TOJSG$, galime gauti ne optimalius svorius arba išviso nepaskaičiuoti $TOJSG$, dėl to kad neegzistuoja skaičiuojamos matricos atvirkštinė. Kadangi mes matricos atvirkštinę skaičiuojame su tikimybių reikšmėmis, mes atvirkštinę gausime beveik visada. Taip pat NT jungiant $Perc$ ar $NPerc$ nebuvo gauti geresni rezultatai nei jungiant neapmokomais būdais.

Vidutiniškai ir blogai klasifikuojantys NT – panašiai kaip ir su neapmokomais NT jungimo būdais, apmokomi NT jungimo būdai grąžina geresnius rezultatus nei vienas geriausias NT. Palyginus apmokomus ir neapmokomus NT jungimo būdus, 7 kartus iš 8 apmokomi NT jungimo būdai buvo pranašesni. Kombinuojant NT klasifikavimas pagerėja nuo 1% iki 60% palyginus su geriausiu neapmokomu jungimo būdu. Su „Mėnuliai“ duomenų imtimi apmokyti vidutiniški NT jungiami apmokomais NT būdais buvo mažiau tikslesni nei neapmokomi NT jungimo būdai. Labiausiai išsiskyrė su „Ratai“ duomenų imtimi apmokyti blogiausi NT, kurių NT junginio tikslumas pasiekė 93%, kai geriausio NT tikslumas yra 55%, o geriausio neapmokomo NT jungimo tikslumas 58%. Toks pagerėjimas buvo pasiektas pasinaudojant $NPerc$ ir $Perc$, vis dėlto, reikėtų nepamiršti, kad norint naudoti šiuos jungimo būdus, reikia papildomo laiko jų apmokymui.

Tarpusavyje lyginant visus realizuotus $TOJSG$ variantus, negalime išskirti vieno geriausio algoritmo. 5 kartus iš 8 $NPerc + TOJS$ arba $Perc + TOJS$ sėkmingai atnaujino TOJS svorius taip, kad tikslumas jungiant NT padidėtų. 1 kart iš 8 $TOJSG$ arba $NPerc + TOJS$ arba $Perc + TOJS$ sujungė NT geriau nei $NPerc$ arba $Perc$. 5 kart iš 8 $TOJSG$ arba $NPerc + TOJS$ arba $Perc + TOJS$ sujungė NT blogiau nei $NPerc$ arba $Perc$. Jungiant vidutiniškai apmokytus NT, $TOJSG$ neaplenkė neapmokomų NT jungimo būdų. $TOJSG$ aplenkė kitus neapmokomus NT jungimo būdus tik jungiant blogiausius rezultatus grąžinančius NT, tačiau lyginant $TOJSG$ su $Perc$ ar $NPerc$ jis nebuvo tikslesnis.

TOJS ir perceptronas inicializuotas TOJS pasirodė gerai tik ant „Gauso“ duomenų imties t.y. tuo teoriškai „idealiu“ atveju, kuriam TOJS ir buvo sukurtas. Taigi, realaus pasaulio duomenų

„negausiškumas“ yra pakankamas, kad gausiniu pasiskirstymu paremti teoriniai modeliai nepadeda geriau apmokyti neparametrinio klasifikatoriaus tokio kaip perceptronas.

Gauti rezultatai: Jungiant geriausiai apmokytus NT, geriausiai yra rinktis vieną tiksliausią NT. Kai jungiame ne geriausiai apmokytus NT, gauname, kad tam tikras jungimo metodas yra geresnis nei geriausias NT. Apmokomi NT jungimo būdai yra pranašesni už neapmokomus NT jungimo būdus, tačiau jie reikalauja papildomo mokymo laiko. Geriausiai ir vidutiniškai apmokytus NT jungiant $TOJSG$ neparodė pranašumo prieš kitus NT jungimo būdus. $TOJSG$ aplenkė neapmokomus jungimo būdus kai jungiami NT buvo blogiausiai apmokyti.

Kadangi NT grupės buvo sudarytos surūšiavus atitinkamos imties NT pagal **V** tikslumą, tiksliausiai klasifikuojantys NT yra labiau prisitaikę prie **V** ir tie NT yra panašūs, tai galimai paaiškina rezultatų skirtumą tarp geriausiai klasifikuojančių NT ir kitų NT. Šitame poskyryje gauti rezultatai negalėtų apsakyti bendrų tendencijų, todėl norint bendriau nusakyti NT jungimo tendencijas buvo atlikti papildomi bandymai.

5.4.2. Bandymas 2

1 bandyme gauti rezultatai neužtikrina rezultatų bendrumo, todėl atliksime papildomą bandymą. 1 bandyme gauti rezultatai išsiskyrė į 2 grupes: tai geriausiai apmokyti NT ir vidutiniškai ir blogai apmokyti NT. Pirmu atveju, kai jungiame NT, geriausiai yra naudoti vieną geriausią NT, o kitu atveju apmokomus NT jungimo būdus. Kadangi 1 bandyme NT buvo pasirinkti neatsitiktinai ir pasirinktas nedidelis jungiamų NT kiekis, tai paanalizuosime NT jungimo tendencijas, kai NT pasirinksime atsitiktinai.

Tikslas: Gauti apibendrintus rezultatus jungiant tam tikros architektūros įvairius NT.

Numanomas rezultatas: Atsitiktinai pasirenkant ir jungiant NT, apmokomi NT jungimo būdai turėtų būti pranašesni už kitus jungimo būdus.

Aprašymas ir gautų rezultatų analizė:

Atsitiktinai pasirenkame duomenų imtį, NT kiekį ir pačius NT. Vykdant šitą bandymą, nebuvo imami NT, kurių tikslumas buvo mažesnis nei 0.5. Bendrai šitame bandyme buvo panaudota 1253 NT. Bandymas vykdytas 100 kartų. Bandyme panaudojamos visos duomenų imtys, kurios gali būti suklasifikuotos į 2 klases: *Ratai, Mėnuliai, Gauso, Vėžys*.

Palyginsime neapmokomus, apmokomus ir geriausią klasifikatorių. Rezultatai pateikiami 4 lentelėje. „Apmok.“ – tai NT jungimas apmokomais NT jungimo būdais. „Neapmok.“ – tai NT jungimas neapmokomais NT jungimo būdais. „Geriau.“ – tai geriausio NT rezultatas.

4 lentelė. Apmokomų, neapmokomų ir geriausio NT jungimo palyginimas

	Apmok. < Geriau.	Apmok. = Geriau.	Apmok. > Geriau.	Suma
Apmok. > Neapmok.	16	20	22	58
Apmok. = Neapmok.	9	24	6	39
Apmok. < Neapmok.	1	2	0	3
	26	46	28	100

Palyginus „Apmok.“ ir „Neapmok.“, tai 58 atvejais „Apmok.“ sujungė NT geriau nei „Neapmok.“. 39 kartus nebuvo skirtumo koki jungimo būdą rinktis. 3 atvejais, „Neapmok.“ aplenkė „Apmok.“. Palyginus „Apmok. > Neapmok.“ su „Apmok. ? Geriau.“, tai „Geriau.“ 16 kartų grąžino geresni rezultatą nei „Apmok.“, o visais likusiais atvejais „Apmok.“ buvo neblogesnis nei „Geriau.“. Žiūrint bendrai, „Apmok.“ yra geriausias NT jungimo būdo tipas, nes jis arba grąžins geresnį rezultatą arba neblogesnį nei „Neapmok.“ ar „Geriau.“.

Jeigu lyginsime TOJS ir $Perc + TOJS$ ar $NPerc + TOJS$, tai gauname: $Perc + TOJS$ ar $NPerc + TOJS$ nepavyko atnaujinti TOJS 59 kartus, $Perc + TOJS$ ar $NPerc + TOJS$ pavyko atnaujinti TOJS 41 kartą. Kaip matome kažkuriam perceptronui pavyko 41 kartą atnaujinti TOJS taip, kad padidėtų tikslumas. Lentelėje 5 pateikiame atnaujintų TOJS tikslumo padidėjimo procentais statistiką.

5 lentelė. Atnaujintų TOJS tikslumo padidėjimas procentais

Min	Max	Avg	Median
0.90%	7.55%	1.79%	1.72%

Kaip matome, vidutiniškai TOJS buvo atnaujinti taip, kad NT kombinavimo tikslumas padidėjo 1.79%.

Jeigu palyginsime $TOJSG$ tarpusavyje, gausime: TOJS buvo tikslesnis nei kiti 2 kartus, TOJS-A buvo tikslesnis nei kiti 5 kartus, TOJS-BP buvo tikslesnis nei kiti 3 kartus, TOJS-ABP buvo tikslesnis nei kiti 1 kartą. Kaip matome, iš 100 bandymų jokia funkcija vienareikšmiškai neaplenkė kitos. Jeigu sušvelninsime sąlygą, gausime: TOJS buvo tikslesnis arba lygus nei kiti 80 kartų, TOJS-A buvo tikslesnis arba lygus nei kiti 40 kartų, TOJS-BP buvo tikslesnis arba lygus nei kiti 81 kartą, TOJS-ABP buvo tikslesnis arba lygus nei kiti 40 kartų. Kaip matome, nors ir joks $TOJSG$ vienareikšmiškai nėra pranašesnis, bet jeigu iškelsime sąlygą, kad jis būtų neblogesnis nei kiti, tai TOJS ir TOJS-BP mažiausiai 80 kartų buvo neblogesnis. Tai dalinai gali paaiškinti TOJS-A ir TOJS-ABP funkcijos, kurios naudoja vieno NT atsakymus kaip poslinkį, taip silpninant bendrą tikslumą.

Jeigu palyginsime $NPerc + TOJS$ arba $Perc + TOJS$ ir $NPerc$ arba $Perc$ gausime: $NPerc + TOJS$ arba $Perc + TOJS$ buvo mažiau tikslūs nei $NPerc$ arba $Perc$ 53 kartus, $NPerc + TOJS$ arba $Perc + TOJS$ buvo tikslesnis nei $NPerc$ arba $Perc$ 5 kartus, $NPerc + TOJS$ arba $Perc + TOJS$ buvo lygus $NPerc$ arba $Perc$ 42 kartus. Kaip matome, TOJS inicializuotas perceptronas tik 5 kartus buvo tikslesnis nei perceptronas inicializuotas atsitiktiniais svoriais.

Jeigu palyginsime TOJS su neapmokomais NT jungimo būdais, gausime: TOJS buvo mažiau tikslūs nei neapmokomi NT jungimo būdai 47 kartus, TOJS buvo lygus neapmokomiems NT jungimo būdams 31 kartą, TOJS buvo tikslesnis nei neapmokomi NT jungimo būdai 22 kartus. Kaip matome neapmokomi NT jungimo būdai buvo neblogesni nei TOJS.

Gauti rezultatai: Žiūrint bendrai, geriausiai yra jungti NT su apmokomais NT jungimo būdais, nes jie arba grąžins geresnį rezultatą arba neblogesnį nei neapmokomi jungimo būdai ar

geriausias klasifikatorius. Jokia konkreti $TOJSG$ skaičiavimo funkcija nėra griežtai pranašesnė už likusias $TOJSG$. TOJS inicializuotas perceptronas 41 kartą iš 100 atnaujino svorius vidutiniškai padidinant NT junginio tikslumą 1.79%. TOJS inicializuoto perceptrono panaudojimas neparodė pranašumų prieš atsitiktinais svoriais inicializuotą perceptroną. Neapmokomi NT jungimo būdai buvo neblogesni nei TOJS. Atsižvelgiant į gautus rezultatus, šiame bandyme gauti rezultatai neparodė, kad TOJS ar perceptronas inicializuotas TOJS yra pranašesnis už kitus NT jungimo būdus.

5.4.3. Bandymas 3

1 bandyme gavome, kad $TOJSG$ buvo pranašesni už neapmokomus NT jungimo būdus jungiant tik blogiausiai apmokytus NT, o tai yra 4 kartus iš 12. 2 bandyme gavome, kad $TOJSG$ buvo pranašesni už neapmokomus NT jungimo būdus 22 kartus iš 100. Manome, kad tokius rezultatus įtakojo NT skirtingumas. Norint tai patikrinti, šio bandymo metu pabandysime padidinti NT skirtingumą 2 būdais:

1. Imsime NT kurių $\mathbf{V}$ tikslumas τ tenkina sąlygą $0.5 > \tau > 0.8$;
2. Imsime NT iš visų apmokytų NT architektūrų;

Tikslas: Patikrinti skirtingumo įtaką jungiant NT, kai lyginame $TOJSG$ ir neapmokomus NT jungimo būdus.

Numanomas rezultatas: NT skirtingumas turėtų padidėti ir $TOJSG$ jungiami NT bus pranašesni už neapmokomus NT jungimo būdus.

Aprašymas ir gautų rezultatų analizė:

Atsitiktinai pasirenkame duomenų imtį, NT kiekį ir pačius NT. Vykdant šitą bandymą, buvo imami NT, kurių $\mathbf{V}$ tikslumas τ tenkina sąlygą $0.5 > \tau > 0.8$. Buvo imami NT iš visų apmokytų NT architektūrų. Bendrai šitame bandyme buvo panaudota 1106 NT. Bandymas vykdytas 100 kartų. Bandyme panaudojamos visos duomenų imtys kurios gali būti suklasifikuotos į 2 klases: *Ratai, Mėnuliai, Gauso, Vėžys*.

Palyginsime neapmokomus, apmokomus ir geriausią klasifikatorių. Rezultatai pateikiami 6 lentelėje. „Apmok.“ – tai NT jungimas apmokomais NT jungimo būdais. „Neapmok.“ – tai NT jungimas neapmokomais NT jungimo būdais. „Geriau.“ – tai geriausio NT rezultatas.

6 lentelė. Apmokomų, neapmokomų ir geriausio NT jungimo palyginimas

	Apmok. < Geriau.	Apmok. = Geriau.	Apmok. > Geriau.	Suma
Apmok. > Neapmok.	0	0	100	100
Apmok. = Neapmok.	0	0	0	0
Apmok. < Neapmok.	0	0	0	0
	0	0	100	100

Kaip matome, „Apmok.“ vienareikšmiškai aplenkė „Neapmok.“ ir „Geriau.“.

Jeigu lyginsime TOJS ir $Perc + TOJS$ ar $NPerc + TOJS$, tai gauname: $Perc + TOJS$ ar $NPerc + TOJS$ nepavyko atnaujinti TOJS 54 kartus, $Perc + TOJS$ ar $NPerc + TOJS$ pavyko atnaujinti TOJS 46 kartus. Kaip matome, tam tikram perceptronui pavyko 46 kartus atnaujinti TOJS taip, kad padidėtų tikslumas. Lentelėje 7 pateikiame atnaujintų TOJS tikslumo padidėjimo procentais statistiką.

7 lentelė. Atnaujintų TOJS tikslumo padidėjimas procentais

Min	Max	Avg	Median
0.91%	9.80%	2.60%	1.92%

Kaip matome, vidutiniškai TOJS buvo atnaujinti taip, kad NT kombinavimo tikslumas padidėjo 2.60%.

Jeigu palyginsime $TOJSG$ tarpusavyje, gausime: TOJS buvo tikslesnis nei kiti 28 kartus, TOJS-A buvo tikslesnis nei kiti 7 kartus, TOJS-BP buvo tikslesnis nei kiti 10 kartų, TOJS-ABP buvo tikslesnis nei kiti 0 kartų. Kaip matome, iš 100 bandymų TOJS funkcija parodė pranašumą prieš kitas funkcijas. Jeigu sušvelninsime sąlygą, gausime: TOJS buvo tikslesnis arba lygus nei kiti 81 kartą, TOJS-A buvo tikslesnis arba lygus nei kiti 10 kartų, TOJS-BP buvo tikslesnis arba lygus nei kiti 62 kartus, TOJS-ABP buvo tikslesnis arba lygus nei kiti 3 kartus. Kaip matome, TOJS funkcija buvo pranašesnė už kitas funkcijas.

Jeigu palyginsime $NPerc + TOJS$ arba $Perc + TOJS$ ir $NPerc$ arba $Perc$ gausime: $NPerc + TOJS$ arba $Perc + TOJS$ buvo mažiau tikslūs nei $NPerc$ arba $Perc$ 54 kartus, $NPerc + TOJS$ arba $Perc + TOJS$ buvo tikslesnis nei $NPerc$ arba $Perc$ 12 kartų, $NPerc + TOJS$ arba $Perc + TOJS$ buvo lygus $NPerc$ arba $Perc$ 34 kartus. Kaip matome, TOJS inicializuotas perceptronas tik 12 kartų buvo tikslesnis nei perceptronas inicializuotas atsitiktiniais svoriais.

Jeigu palyginsime TOJS su neapmokomais NT jungimo būdais, gausime: TOJS buvo mažiau tikslūs nei neapmokomi NT jungimo būdai 10 kartų, TOJS buvo lygus neapmokomiems NT jungimo būdams 8 kartus, TOJS buvo tikslesnis nei neapmokomi NT jungimo būdai 82 kartus. Kaip matome, TOJS buvo pranašesnis už neapmokomus NT jungimo būdus.

Gauti rezultatai: Žiūrint bendrai, geriausiai yra jungti NT su apmokomais NT jungimo būdais, nes jie grąžins geresnį rezultatą nei neapmokomi jungimo būdai ar geriausias klasifikatorius. TOJS funkcija yra pranašesnė už likusias $TOJSG$ funkcijas. TOJS inicializuotas perceptronas 46 kartus iš 100 atnaujino svorius vidutiniškai padidinant NT junginio tikslumą 2.60%. TOJS inicializuoto perceptrono panaudojimas neparodė pranašumų prieš atsitiktinais svoriais inicializuotą perceptroną. TOJS buvo pranašesnis už neapmokomus NT jungimo būdus. Atsižvelgiant į gautus rezultatus, šiame bandyme gauti rezultatai parodė, kad TOJS yra pranašesnis už neapmokomus NT jungimo būdus.

5.4.4. Bandymas 4

Kadangi 3 bandyme analizavome tik imtis, kurios gali būti klasifikuojamos į 2 klases, praplečiame 3 bandymą jį vykdydami tik su imtimis, kurios gali būti klasifikuojamos į daugiau nei 2 klases. Bandymas atliekamas atskirai nuo 3 bandymo, nes TOJS funkciją ir perceptrono algoritmą pritaikėme daugiaklasiui klasifikavimui.

Tikslas: Patikrinti ar daugiaklasiui klasifikavimui pritaikytas TOJS grąžina skirtingus rezultatus nei į 2 klases klasifikuojantys TOJS.

Numanomas rezultatas: Gauti rezultatai dalinai skirsis nuo 3 bandyme gautų rezultatų.

Aprašymas ir gautų rezultatų analizė:

Atsitiktinai pasirenkame duomenų imtį, NT kiekį ir pačius NT. Vykdant šitą bandymą, buvo imami NT, kurių **V** tikslumas τ tenkina sąlygą $0.5 > \tau > 0.8$. Buvo imami NT iš visų apmokytų NT architektūrų. Bendrai šitame bandyme buvo panaudota 4767 NT. Bandymas vykdytas 100 kartų. Bandyme panaudojamos visos duomenų imtys kurios gali būti suklasifikuotos į daugiau nei 2 klases: *Suktinis, S raidė, Iris, Vynas, Raudono vyno kokybė, Balto vyno kokybė*. Jungiant NT apmokomais NT jungimo būdais, NT atsakymai buvo abstraktaus tipo (imamas klasės numeris kaip atsakymas).

Palyginsime neapmokomus, apmokomus ir geriausią klasifikatorių. Rezultatai pateikiami 6 lentelėje. „Apmok.“ – tai NT jungimas apmokomais NT jungimo būdais. „Neapmok.“ – tai NT jungimas neapmokomais NT jungimo būdais. „Geriau.“ – tai geriausio NT rezultatas.

8 lentelė. Apmokomų, neapmokomų ir geriausio NT jungimo palyginimas

	Apmok. < Geriau.	Apmok. = Geriau.	Apmok. > Geriau.	Suma
Apmok. > Neapmok.	3	1	58	62
Apmok. = Neapmok.	0	3	4	7
Apmok. < Neapmok.	2	1	28	31
	5	5	90	100

Palyginus „Apmok.“ ir „Neapmok.“, tai 62 atvejais „Apmok.“ sujungė NT geriau nei „Neapmok.“. 7 kartus nebuvo skirtumo kokį jugimo būdą rinktis. 31 atveju „Neapmok.“ aplenkė „Apmok.“. Palyginus „Apmok. > Neapmok.“ su „Apmok. ? Geriau.“, tai „Geriau.“ 90 kartų grąžino mažiau tikslų rezultatą nei „Apmok.“. Žiūrint bendrai, „Apmok.“ yra geriausias NT jungimo būdo tipas, nes jis arba grąžins geresnį rezultatą arba neblogesnį nei „Neapmok.“ ar „Geriau.“.

Jeigu lyginsime TOJS ir $Perc + TOJS$ ar $NPerc + TOJS$, tai gauname: $Perc + TOJS$ ar $NPerc + TOJS$ nepavyko atnaujinti TOJS 20 kartų, $Perc + TOJS$ ar $NPerc + TOJS$ pavyko atnaujinti TOJS 80 kartų. Kaip matome, tam tikram perceptronui pavyko 80 kartų atnaujinti TOJS taip, kad padidėtų tikslumas. Lentelėje 9 pateikiame atnaujintų TOJS tikslumo padidėjimo procentais statistiką.

9 lentelė. Atnaujintų TOJS tikslumo padidėjimas procentais

Min	Max	Avg	Median
0.18%	28.28%	6.54%	4.26%

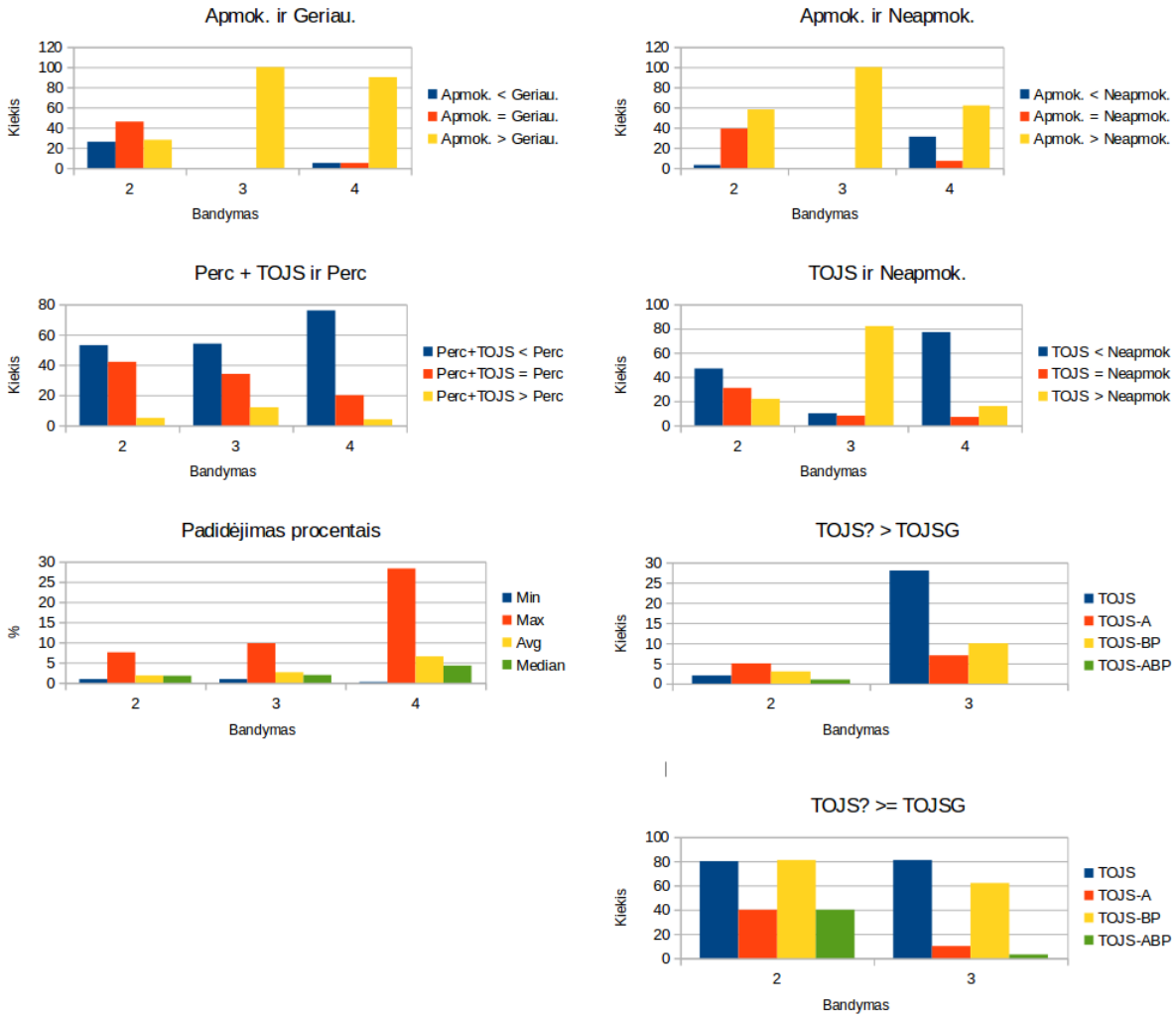
Kaip matome, vidutiniškai TOJS buvo atnaujinti taip, kad NT kombinavimo tikslumas padidėjo 6.54%.

Jeigu palyginsime $NPerc + TOJS$ arba $Perc + TOJS$ ir $NPerc$ arba $Perc$ gausime: $NPerc + TOJS$ arba $Perc + TOJS$ buvo mažiau tikslus nei $NPerc$ arba $Perc$ 76 kartus, $NPerc + TOJS$ arba $Perc + TOJS$ buvo tikslesnis nei $NPerc$ arba $Perc$ 4 kartus, $NPerc + TOJS$ arba $Perc + TOJS$ buvo lygus $NPerc$ arba $Perc$ 20 kartų. Kaip matome, TOJS inicializuotas perceptronas tik 4 kartus buvo tikslesnis nei perceptronas inicializuotas atsitiktiniais svoriais.

Jeigu palyginsime TOJS su neapmokomais NT jungimo būdais, gausime: TOJS buvo mažiau tikslus nei neapmokomi NT jungimo būdai 77 kartus, TOJS buvo lygus neapmokomiems NT jungimo būdams 7 kartus, TOJS buvo tikslesnis nei neapmokomi NT jungimo būdai 16 kartų. Kaip matome, TOJS buvo mažiau tikslus nei neapmokomi NT jungimo būdai. Šio bandymo metu buvo analizuojamos ir dvi 1 bandymo metu išsiskyrusios duomenų imtys: raudono ir balto vyno kokybė. Buvo manoma, kad tai įtakojo tokius rezultatus, tačiau išėmus šias imtis iš analizės, gavome tokias pačias išvadas: neapmokomi NT jungimo būdai NT jungia geriau nei TOJS.

Gauti rezultatai: Žiūrint bendrai, geriausiai yra jungti NT su apmokomais NT jungimo būdais, nes jie grąžins neblogesnį rezultatą nei neapmokomi jungimo būdai ar geriausias klasifikatorius. TOJS inicializuotas perceptronas 80 kartų iš 100 atnaujino svorius vidutiniškai padidinant NT junginio tikslumą 6.54%. TOJS inicializuoto perceptrono panaudojimas neparodė pranašumų prieš atsitiktinais svoriais inicializuotą perceptroną. Šiame bandyme gauti rezultatai parodė, kad TOJS nėra tikslesnis už neapmokomus NT jungimo būdus.

5.4.5. Bandymų apibendrinimas



6 pav. Bandymų rezultatai. x ašyje bandymo numeris, y ašyje arba bandymo metu gautas įvykio kiekis arba procentai.

Bendru atveju 2,3 ir 4 bandymai patvirtina, kad tiksliau klasifikuosime kai NT bus jungiami apmokomais NT jungimo būdais, palyginus su neapmokomais NT jungimo būdais ir geriausiu NT (6 pav. „Apmok. ir Geriau.“ ir „Apmok ir Neapmok.“). 1 bandyme gavome, kad palyginus TOJS su neapmokomais NT jungimo būdais, TOJS parodė gerus rezultatus, kai jungėme blogiausiai klasifikuojančius NT. Dėl mūsų NT apmokymo būdo, laikome kad blogiausiai apmokyti NT bus labiausiai skirtingi. 2 bandyme gavome, kad TOJS retai aplenkdavo neapmokomus NT jungimo būdus. 3 bandyme siekėme padidinti NT skirtingumą, neimdami NT su tikslumu didesniu nei 0.8. 3 bandyme gavome, kad TOJS dažnai aplenkdavo neapmokomus NT jungimo būdus. Galime teigti, kad jungiant skirtingesnius NT, TOJS aplenkia neapmokomus NT jungimo būdus (6 pav. „TOJS ir Neapmok.“). Šito fakto nustatymas yra svarbus, nes TOJS apskaičiavimo greitis palyginus su perceptrono apmokymo greičiu, yra daug kartų mažesnis. 2 bandyme joks $TOJSG$ nebuvo vienareikšmiškai pranašesnis. 3 bandyme padidintas skirtingumas išreiškė TOJS funkcijos pranašumą prieš likusias $TOJSG$ funkcijas (6 pav. „TOJS? > TOJSG“ ir „TOJS? >= TOJSG“). 3 bandyme padidintas skirtingumas nepakeitė to, kad TOJS inicijuotas perceptronas

gali dar pagerinti junginio tikslumą (6 pav. „Padidėjimas procentais“), tačiau tiksliau, vis dėlto, galime NT jungti su perceptronu inicijuotu atsitiktiniais svoriais (6 pav. „Perc+TOJS ir Perc“); 2, 3 ir 4 bandyme gauti rezultatai iš esmės nesiskyrė, o tai nurodo, kad padidintas skirtingumas ar klasių kiekis neįtakoja rezultato. 4 bandyme gauti rezultatai parodė, kad TOJS nėra tikslesnis už neapmokomus NT jungimo būdus.

6. Tyrimo išvados

Šiame darbe buvo atlikta:

1. Išanalizuota literatūra.
2. Sukurti ir apmokyti NT.
3. Realizuota dalis NT kombinavimo funkcijų.
4. Realizuotas perceptrono algoritmas.
5. Realizuotos keturios teoriškai optimalių svorių skaičiavimo funkcijos.
6. Viena teoriškai optimalių svorių skaičiavimo funkcija pritaikyta perceptrono svoriams inicializuoti.
7. Viena teoriškai optimalių svorių skaičiavimo funkcija pritaikyta klasifikavimo uždaviniui spręsti, kai imtis turi daugiau nei 2 klases.
8. Atlikti bandymai patikrinantys TOJS panaudojamumą.

Atlikus tyrimą, gautos tokios išvados:

1. **Pagrindinė išvada**: Realaus pasaulio duomenų „negausiškumas“ yra pakankamas, kad gausiniu pasiskirstymu paremti teoriniai modeliai nepadeda geriau apmokyti neparametrinio klasifikatoriaus tokio kaip perceptronas.
2. Neparametrinis klasifikatorius toks kaip perceptronas gali tam tikrais atvejais ištaisyti teorinio modelio prielaidų neatitikimus. Tai patvirtina gauti rezultatai. Vis dėlto, atsitiktinais svoriais inicijuotas perceptronas sugebėjo tiksliau jungti NT.
3. Gausiniu pasiskirstymu paremti teoriniai modeliai sugeba tiksliau jungti skirtingus NT lyginant juos su neapmokomais NT jungimo būdais. Tai reiškia, kad galimai ir kitais pasiskirstymais paremti teoriniai modeliai gali būti panaudoti NT jungime.

Literatūra

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo ir k.t. Tensorflow: large-scale machine learning on heterogeneous distributed systems. 2015. URL: `http://download.tensorflow.org/paper/whitepaper2015.pdf`.

[ADF+15] Muhammad AO Ahmed, Luca Didaci, Giorgio Fumera ir Fabio Roli. An empirical investigation on the use of diversity for creation of classifier ensembles. *International workshop on multiple classifier systems*. Springer, 2015, p.p. 206–219.

[APR07] Javed A Aslam, Raluca A Popa ir Ronald L Rivest. On estimating the size and confidence of a statistical audit. *Evt*, 7:8, 2007.

[BSE+97] Jon Atli Benediktsson, Johannes R Sveinsson, Okan K Ersoy ir Philip H Swain. Parallel consensual neural networks. *Ieee transactions on neural networks*, 8(1):54–64, 1997.

[CCA+09] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos ir José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision support systems*, 47(4):547–553, 2009.

[CTY09] Huanhuan Chen, Peter Tiňo ir Xin Yao. Predictive ensemble pruning by expectation propagation. *Ieee transactions on knowledge and data engineering*, 21(7):999–1013, 2009.

[Die97] Thomas G Dietterich. Machine-learning research. *Ai magazine*, 18(4):97, 1997.

[Dui02] Robert PW Duin. The combining classifier: to train or not to train? *Pattern recognition, 2002. proceedings. 16th international conference on*. Tom. 2. IEEE, 2002, p.p. 765–770.

[GM88] Alan S Gevins ir NH Morgan. Applications of neural-network (nn) signal processing in brain research. *Ieee transactions on acoustics, speech, and signal processing*, 36(7):1152–1161, 1988.

[Gol89] David E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st leid., 1989. ISBN: 0201157675.

[GVC05] Pablo M Granitto, Pablo F Verdes ir H Alejandro Ceccatto. Neural network ensembles: evaluation of aggregation algorithms. *Artificial intelligence*, 163(2):139–162, 2005.

[Hay09] Simon Haykin. *Neural networks and learning machines*. Tom. 3. Pearson Upper Saddle River, NJ, USA: 2009.

[Has93] Sherif Hashem. Optimal linear combinations of neural networks1. Disertacija. Purdue University, 1993.

[HK00] Jakob V Hansen ir Anders Krogh. A general method for combining predictors tested on protein secondary structure prediction. *Artificial neural networks in medicine and biology*, p.p. 259–264. Springer, 2000.

[YI08] Xin Yao ir Md Monirul Islam. Evolving artificial neural network ensembles. *Ieee computational intelligence magazine*, 3(1):31–42, 2008.

[KB14] Diederik P Kingma ir Jimmy Ba. Adam: a method for stochastic optimization. *Arxiv preprint arxiv:1412.6980*, 2014.

[KF15] Andrej Karpathy ir Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *Proceedings of the ieee conference on computer vision and pattern recognition*, 2015, p.p. 3128–3137.

[Koh90] Teuvo Kohonen. The self-organizing map. *Proceedings of the ieee*, 78(9):1464–1480, 1990.

[Kun02] Ludmila I Kuncheva. A theoretical study on six classifier fusion strategies. *Ieee transactions on pattern analysis and machine intelligence*, 24(2):281–286, 2002.

[Kun14] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2014.

[Kun16] Ludmila Kuncheva. Getting lost in the wealth of classifier ensembles? *Icpram*, 2016, p. 7.

[KV+95] Anders Krogh, Jesper Vedelsby ir k.t. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7:231–238, 1995.

[KW03] Ludmila I Kuncheva ir Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

[Lam00] Louisa Lam. Classifier combinations: implementations and theoretical issues. *International workshop on multiple classifier systems*. Springer, 2000, p.p. 77–86.

[Lic13] M. Lichman. UCI machine learning repository. 2013. URL: `http://archive.ics.uci.edu/ml`.

[MHS+13] H Brendan McMahan, Gary Holt, David Sculley, Michael Young ir k.t. Ad click prediction: a view from the trenches. *Proceedings of the 19th acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2013, p.p. 1222–1230.

[Min01] Thomas P Minka. Expectation propagation for approximate bayesian inference. *Proceedings of the seventeenth conference on uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2001, p.p. 362–369.

[MSJ+12] João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge ir Jorge Freire De Sousa. Ensemble approaches for regression: a survey. *Acm computing surveys (csur)*, 45(1):10, 2012.

[Pau01] Anne Magaly de Paula Canuto. Combining neural networks and fuzzy logic for applications in character recognition. Disertacija. University of Kent at Canterbury, 2001.

[RK96] Søren Kamaric Riis ir Anders Krogh. Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of computational biology*, 3(1):163–183, 1996.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RP05] Romesh Ranawana ir Vasile Palade. A neural network based multi-classifier system for gene identification in dna sequences. *Neural computing & applications*, 14(2):122–131, 2005.

[RP06] Romesh Ranawana ir Vasile Palade. Multi-classifier systems: review and a roadmap for developers. *International journal of hybrid intelligent systems*, 3(1):35–61, 2006.

[RRD15] Fred Richardson, Douglas Reynolds ir Najim Dehak. Deep neural network approaches to speaker and language recognition. *Ieee signal processing letters*, 22(10):1671–1675, 2015.

[SAA13] Symone Soares, Carlos Henggeler Antunes ir Rui Araújo. Comparison of a genetic algorithm and simulated annealing for automatic neural network ensemble development. *Neurocomputing*, 121:498–511, 2013.

[San11] Roli Fabio Sansone Carlo Kittler Josef. *Multiple classifier systems. 10th international workshop, mcs 2011, naples, italy, june 15–17, 2011. proceedings*. Tom. 6713, 2011-06.

[SB00] Holger Schwenk ir Yoshua Bengio. Boosting neural networks. *Neural computation*, 12(8):1869–1887, 2000.

[Sch15a] Jürgen Schmidhuber. Deep learning in neural networks: an overview. *Neural networks*, 61:85–117, 2015.

[Sch15b] Kittler Josef Schwenker Friedhelm Roli Fabio. *Multiple classifier systems. 12th international workshop, mcs 2015, günzburg, germany, june 29 - july 1, 2015, proceedings*. Tom. 9132, 2015-06.

[Sha12] Amanda JC Sharkey. *Combining artificial neural nets: ensemble and modular multi-net systems*. Springer Science & Business Media, 2012.

[Tip01] Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001.

[Ued00] Naonori Ueda. Optimal linear combination of neural networks for improving classification performance. *Ieee transactions on pattern analysis and machine intelligence*, 22(2):207–215, 2000.

[Wan03] Nayer Wanas. Feature based architecture for decision fusion. Disertacija. University of Waterloo, 2003.

[WDF15] Ian Walker, Marc Deisenroth ir Aldo Faisal. Deep convolutional neural networks for brain computer interface using motor imagery. *Imperial college of science, technology and medicine department of computing*, 2015.

[WGC14] Michał Woźniak, Manuel Graña ir Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information fusion*, 16:3–17, 2014.

[Woz07] Michal Wozniak. Experiments with trained and untrained fusers. *Innovations in hybrid intelligent systems*, p.p. 144–150. Springer, 2007.

[XKS92] Lei Xu, Adam Krzyzak ir Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *Ieee transactions on systems, man, and cybernetics*, 22(3):418–435, 1992.

[Zho12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[Zho13] Kittler Josef Zhou Zhi-Hua Roli Fabio. *Multiple classifier systems. 11th international workshop, mcs 2013, nanjing, china, may 15-17, 2013. proceedings*. Tom. 7872, 2013-05.

[ZWT02] Zhi-Hua Zhou, Jianxin Wu ir Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1):239–263, 2002.