RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Lietišķo datorsistēmu institūts

Mākslīgā intelekta un sistēmu inženierijas katedra

Rinalds Daniels Pikše

bakalaura akadēmiskās studiju programmas ”Datorsistēmas”
students, stud. apl. nr. 171RDB359

BEIESA NEIRONU TĪKLU UN MONTE KARLO CAURKRITES METOŽU STABILITĀTES SALĪDZINOŠĀ ANALĪZE

BAKALAURA DARBS

Zinātniskais vadītājs PhD.sc.comp., pētnieks
Ēvalds Urtāns

RĪGA 2023

RĪGAS TEHNISKĀ UNIVERSITĀTE

DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE

Lietišķo datorsistēmu institūts

Mākslīgā intelekta un sistēmu inženierijas katedra

Bakalaura darba izpildes lapa

Noslēguma darba autors:

students Rinalds Daniels Pikše _____

(paraksts, datums)

Noslēguma darbs ieteikts aizstāvēšanai:

Zinātniskais vadītājs:

PhD.sc.comp., pētnieks Ēvalds Urtāns _____

(paraksts, datums)

ANOTĀCIJA

Atslēgvārdi: Mašīnmācīšanās, Beiesa metodes, neironu tīkli
Bakalaura darba tips: Moderno risinājumu izpēte

Dziļā mašīnmācīšanās var sasniegt ievērojamus rezultātus dažādos uzdevumos, taču to stabilitāte un precizitāte var būt grūti nosakāma, jo algoritma neironi ir slēpti. Lai atrisinātu šo problēmu, iespējams apskatīt veidus kā apvienot mākslīgo neironu modeļus ar varbūtības teorijas metodēm, lai sniegtu pārliecības mēru līdz ar prognozēto rezultātu. Beiesa statistika piedāvā veidu, kas ļauj noteikt un kvantitatīvi izteikt modeļa pārliecības pakāpi attiecībā uz konkrētiem rezultātiem. Tie cenšas modelēt šo pārliecības mēru gala rezultātā nodrošinot varbūtību sadalījumu, nevis punkta vērtību.

Idejas par Beiesa neironu tīkliem ir strauji attīstījušās kopš 1990. gadiem un ir izgudrotas tādas metodes kā variāciju inference (VI) (variational inference) (Jordan, Ghahramani et al., 1999), Monte karlo caurkrites (MC-D) (Monte Carlo dropout) izmantošana tuvinātai Beiesa inferencei (Gal & Ghahramani, 2015) un Beies caur atpakaļizplatīšanos (BBB) (Bayes by backprop) (Blundell, Cornebise et al., 2015).

Mērķis pētījumam ir salīdzināt, kā pieminētās metodes, atspoguļo pārliecības mēru pieaugošos datu trokšņa apstākļos. Eksperimenti parāda BBB un MC-D algoritmu ir spējīgi precīzāk klasificēt sēņu datu kopas vērtības. Turklāt tiek novērtos, ka visi trīs algoritmu pārliecības mēri ir jūtīgi pret datu troksni apmācības datos. Būtiski, ka visi algoritmi izrādīja izteiktāku jūtību pret izvaddatu troksni nekā ievaddatu troksni.

Bakalaura darbs sastāv no 50 lappusēm, 19 attēliem, 10 tabulām un 12 pielikumiem. Izmantotās literatūras un avotu sarakstā ir ietverti 49 avoti.

ABSTRACT

Atslēgvārdi: Machine learning, Bayesian methods, neural networks
Bachelor thesis type: Research of modern solutions.

Deep learning can achieve great results in many different tasks, though, their stability and accuracy can be hard to estimate as the algorithms neurons are hidden. To solve this issue, one should look at combining the methods of machine learning and probability theory so as to obtain an uncertainty measure together with a predictive result. Bayesian statistics provide a format that allows one to determine and quantify the uncertainty of a neural networks prediction. It aims to model the uncertainty and provide a distribution of values instead of a point value as a result.

Ideas regarding Bayesian neural networks have rapidly developed since the 1990s and there have been many methods developed such as Variational Inference (VI) (Jordan, Ghahramani et al., 1999), Monte Carlo Dropout (MC-D) usage for inference approximation (Gal & Ghahramani, 2015) and Bayes by Backprop (BBB) (Blundell, Cornebise et al., 2015).

The aim of the study is to compare how the aforementioned methods reflect the degree of confidence in increasing data noise conditions. Experiments demonstrate that the BBB and MC-D algorithms are capable of classifying the values of the mushroom dataset with greater precision. Additionally, it is observed that the confidence measures of all three algorithms are sensitive to data noise in training data. Notably, all algorithms displayed a higher sensitivity towards output data noise than input data noise.

The bachelor's thesis is composed of 50 pages, including 19 figures, 10 tables and 12 attachments. The list of references and sources includes 49 entries.

SATURS

IEVADS

Dziļā mašīnmācīšanās ir mašīnmācīšanās apakšnozare, kas izmanto daudz-slāņu mākslīgos neironu tīklus, lai apstrādātu datus un iegūtu no tiem funkcijas, kas spējīgas aprēķināt rezultātus no jauniem datiem. Dziļā mašīnmācīšanās var sasniegt ievērojamus rezultātus dažādos uzdevumos, taču to stabilitāte un precizitāte var būt grūti nosakāma, to algoritmiskās sarežģītības dēļ. Uzticamība modeļa rezultātiem var būt ļoti svarīga gadījumos, kad to pielietošana notiek augsta riska apstākļos - medicīnā, finansēs u.c. Ja cilvēku veselība vai labklājība ir atkarīga no algoritma pieņemtā lēmuma, zināt pārliecību, ar kuru tiek pieņemts lēmums, var būt kritiski svarīgi.

Dziļās mašīnmācīšanās algoritmi var saturēt ļoti lielu skaitu parametru un bieži satur nelinearitātes slāņus, kas tiek izmantoti, lai atrastu nelineāras attiecības datu kopās.. Datu kvalitāte mašīnmācīšanās risinājumos ir ļoti svarīga, jo tā tieši ietekmē apmācītā modeļa spēju veikt atbilstošus paredzējumus par iepriekš neredzētiem datiem - gadījumos, kad apmācības datu skaits ir limitēts, zinot modeļa uzticamību var būt kritiski, lai veiktu lēmumu, balstoties no modeļa sniegtajiem rezultātiem. Lai atrisinātu šīs problēmas, tiek apskatīts veids kā apvienot mākslīgo neironu modeļus ar varbūtības teorijas metodēm, lai sniegtu pārliecības mēru līdz ar prognozēto rezultātu.

Beiesa neironu tīkli ļauj noteikt un kvantitatīvi izteikt modeļa pārliecības pakāpi attiecībā uz konkrētiem rezultātiem. Tie cenšas modelēt šo pārliecības mēru, gala rezultātā nodrošinot varbūtību sadalījumu, nevis punkta vērtību(Gal & Ghahramani, 2015). Idejas par Beiesa neironu tīkliem ir strauji attīstījušās kopš 1990.gada, un ir izgudrotas tādas metodes kā Variāciju Inference (VI) (variational inference) (Jordan, Ghahramani et al., 1999), Monte Karlo caurkrites (Monte Carlo dropout) (MC-D) izmantošana aptuvenai Beiesa inferencei(Gal & Ghahramani, 2015), Beies caur atpakaļizplatīšanos (Bayes by backprop) (BBB) (Blundell, Cornebise et al., 2015) un citas (Jospin, Buntine et al., 2020).

Vairāki pētījumi ir apskatījuši pieejas, kā implementēt Beiesa metodes makslīgo neironu tīkliem, un salīdzinājuši tās (Gal & Ghahramani, 2015), (Staber & Veiga, 2022). Šajā darbā tiek izveidota jauna metode, kā salīdzināt eksistējošos Beiesa neironu tīklu algoritmus, un tiek pētīts, kā šīs metodes salīdzinās viena ar otru modeļu stabilitātes noteikšanā vaur rezultāta ticamības mēra izveidi.

Darba mērķis ir salīdzināt dažādas Beiesa neironu tīklu metodes ar Monte karlo caurkrites metodēm, izmantojot tabulāros datus.

Mērķa sasniegšanai tika izvirzīti sekojošie uzdevumi:

- Aprakstīt 3 esošās Beiesa tīklu metodes:
  - Variāciju inference (Variational Inference)
  - Monte Karlo caurkrite (Monte Carlo Dropout)

  - Beies caur atpakaļizplatīšanos (Bayes by backprop)

- Izstrādāt algoritmus metožu implementācijai python programmēšanas valodā
- Atlasīt tabulārus datus eksperimentālai daļai
- Salīdzināt risinājumus 2 dažādos veidos:
  - Modeļu stabilitātes salīdzināšana, pieaugot datu kopas trokšņainumam
  - Modeļu izveidotās dispersijas salīdzināšana, pieaugot datu kopas trokšņainumam

Bakalaura darba pirmajā daļā tiek apskatīti saistītie pētījumi, kādi jau ir veikti saistībā ar Variāciju Inferenci un Beiesa neironu tīkliem. Tālāk tiek apskatīta metodoloģija, kāda var tikt izmantota vai jau ir bijusi izmantota, lai salīdzinātu apskatītos algoritmus un aprakstītas metodes, kuras izmantotas, lai veiktu salīdzināšanu šajā darbā. Implementācijas sadaļā tiek aprakstīts kā veikti eksperimenti un specifiskā realizācija katram no trim apskatītajiem algoritmiem. Beigās tiek aprakstīti eksperimentu rezultāti, veikti ieteikumi par iespējamiem nākotnes pētījumiem un apkopoti secinājumi, kas izgūti veicot šo pētījumu.

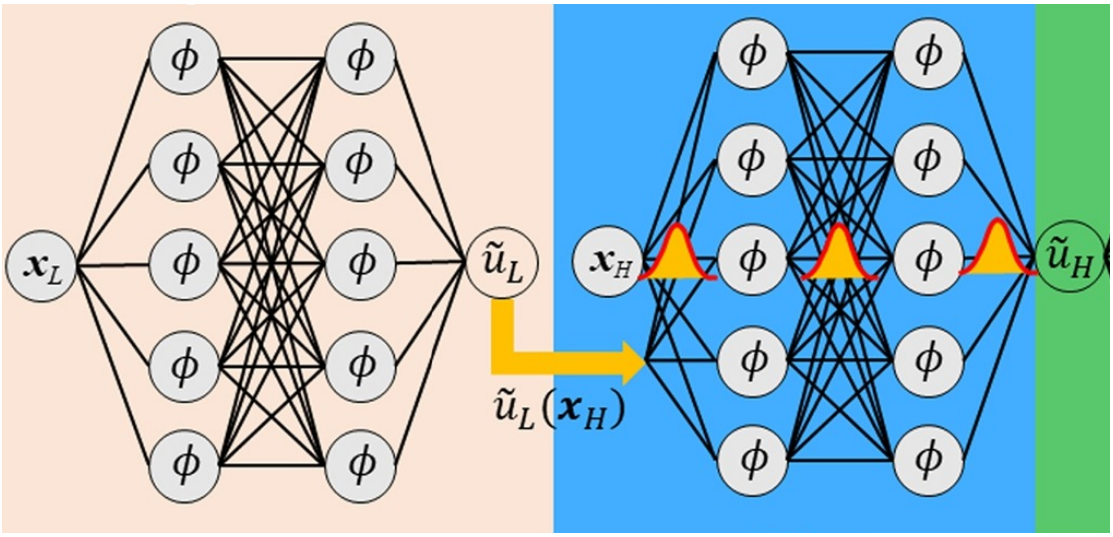
1. SAISTĪTIE PĒTĪJUMI

Beiesa metodes cenšas izmantot varbūtību, lai kvantificētu nenoteiktību secinājumos, kas balstīti uz datu analīzi (Gelman, Carlin et al., 2013). Varbūtību apgalvojumi, ko veic Beiesa metodes ir atkarīgi no novērotajiem datiem - tā tie atšķiras no citam metodēm (Gelman, Carlin et al., 2013). Beiesa inference ir statistiska metode, kas balstās uz Beiesa teorēmu, kas cenšas noteikt varbūtību iespējamību notikumam, ņemot vērā gan iepriekš zināmu informāciju, gan arī jauniegūtos novērojuma datus. Beiesa formula ir sekojoša (Bernardo & Smith, 1994):

$$P(H|D) = \frac{P(D|H) * P(H)}{P(D)} \tag{1.1}$$

kur, $P(D)$ - novērotie dati;
$P(H)$ - sākotnējais varbūtības sadalījums;
$P(D|H)$ - varbūtība H notikumam, ja ir patiess D novērojums);
$P(H|D)$ - varbūtība D notikumam, ja ir patiess H novērojums.

Beiesa neironu tīkli ir mākslīgie neironu tīkli, kas tiek apmācīti izmantojot Beiesa inferenci (Jospin, Buntine et al., 2020). Beiesa neironu tīkli, izmanto sākotnējo varbūtības sadalījumu apmācāmajiem parametriem un cenšas iemācīties gala varbūtību sadalījumu šiem parametriem (Wang & Yeung, 2016). Dažādas pieejas ir radušās, lai tuvinātu šo gala varbūtību, vispārēji tās var iedalīt divās grupās - paraugu atlases metodēs un optimizācijas metodēs (Ai, S. Liu et al., 2021). Atšķirība starp abām pieejām ir tā, ka paraugu atlases metožu gadījumā tiek veidots algoritms, kurš vieniem ienākošiem parametriem atgriezīs mainīgu rezultātu, un, izpildot šo algoritmu atkārtoti, rezultātā izveidojas daudz vērtību, no kurām ir iespējams izveidot vērtību sadalījumu. Kontrastā optimizācijas metodēs vērtību sadalījums tiek ieviests funkciju parametros jeb parametrizēts un šo parametru vērtības tiek apmācītas, lai iegūtu gala vērtību sadalījumu (Jospin, Buntine et al., 2020). Beiesa variāciju inference jau ir daudz aprakstīta un apskatīta (Blei, Kucukelbir et al., 2016; Shridhar, Laumann et al., 2019). Tā tiek uzskatīta par optimizācijas metodi, kas cenšas izveidot vērtību sadalījumu un tad to uzlabot ar dažādām metodēm. Beies caur atpakaļizplatīšanos, arī tiek uzskatīta par optimizācija metodi, taču tas modificē Beiesa inferences pieeju, lai varētu izmantot atpakaļizplatīšanās algoritmu (Jospin, Buntine et al., 2020). Metodes kā Monte Karlo caurkrite strādā pēc paraugu atlases principa - tās izveido modeli, kas atgriež mainīgu rezultātu, kuru atkārtoti izpildot tiek izveidots gala vērtību saraksts, ko iespējams uztvert par vērtību sadalījumu. Šajā darbā tiks padziļināti apskatītas šīs trīs metodes.



1.1. att. Ilustrācija parāda teorētisko atšķirību starp Beiesa neironu tīklu un tradicionāliem neironu tīkliem.(Meng, Babaee et al., 2020)

Ir svarīgi izšķirt starp divu tipu nenoteiktībām - aleatoriskā un epistēmiskā nenoteiktība. Aleatoriska nenoteiktība notver troksni, kas novērojams mērījumos, bet epistēmiskā nenoteiktība notver modeļa izveidoto nenoteiktību. Ir vairāki pētījumi, kas atdala šos nenoteiktības avotus. Pētījums (Kendall & Gal, 2017) parāda, ka aleatoriskā nenoteiktība nesamazinās, samazinot atšķirību starp apmācības datiem un pārbaudes datiem, taču epistemistkā samazinās. Salīdzinot dažādas datu kopas, iespējams novērot korelāciju starp datukopas tīrību un epistēmisko nenoteiktību, bet ne aleatorisko nenoteiktību(Shridhar, Laumann et al., 2018).

1.1. Variāciju Inferences metode

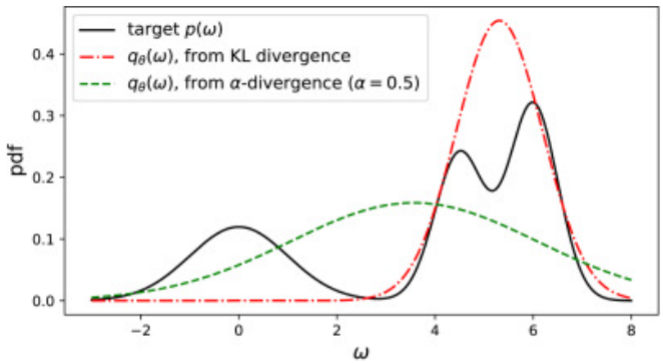
Par inferenci sauc procesu, kur no dotiem parametriem tiek mēģināts iegūt latentos mainīgos. Latentie mainīgie ir netieši mainīgie, kas iegūti no novērojamiem mainīgajiem un kas ietekmē gala rezultātu, tos bieži sauc arī par paslēptajiem (hidden) mainīgajiem. Variāciju inferences mērķis ir veidot tuvinātu gala (posterioro) varbūtību no novērotajiem un latentajiem mainīgajiem (Blei, Kucukelbir et al., 2016).

1995. gadā D. McKay ieviesa veidu, kā aprēķināt gala varbūtības sadalījumu, izmantojot vidējā lauka (MFVI) metodi, kuras ideja ir izmantot katra latentā mainīgā vērtību sadalījumu un apvienot tos, lai nonāktu pie gala vērtību sadalījuma. Gala vērtību sadalījums tiek optimizēts, izmantojot varbūtības sadalījumu tuvināšanas tehnikas, piemēram, Kullback-Leibler (KL) novirzes minimizēšanu, kas mēra divu varbūtību sadalījumu starpību balstoties uz Šanona informācijas teoriju (Jospin, Buntine et al., 2020). Taču KL minimizēšana nav viegli pielietojama, jo mašīnmācīšanās algoritmos ne vienmēr ir piekļuve pie īstā vērtību sadalījuma, tāpēc izmantojot Jensena nevienādību ir iespējams iegūt jaunu optimizējamu mēru, ko sauc par ELBO (Evidence Lower

BOund) (Blei, Kucukelbir et al., 2016). KL novirzes minimizēšana ir ekvivalenta ar ELBO maksimizēšanu taču ELBO funkcijas maksimizēšanai nav nepieciešams zināt īsto vērtību sadalījumu, lai to optimizētu. (Jospin, Buntine et al., 2020).

Lai maksimizētu ELBO vērtību ir var tikt izmantotas vairākas metodes. Gradienta nolaišanās metode, kas ir plaši izmantota dažādiem algoritmiem ir iespējama - variāciju sadalījuma parametru svari tiek izmainīti, lai maksimizētu ELBO vērtību. Stohastiskā variāciju inference (SVI) izmanto paraugu partijas (batches), kas veido varbūtību sadalījumus, kas iteratīvi tiek uzlaboti, izmantojot stohastisko optimizāciju. (Hoffman, Blei et al., 2012) Šī pieeja var būt mērogojamāka, jo ELBO iespējams aprēķināt katrai paraugu partijai, katrā ciklā jeb apmācības iterācijā. (Jospin, Buntine et al., 2020) ADVI ir cita metode, kas cenšas maksimizēt ELBO vērtību pārveidojot inferences problēmu vienotā telpā un tad atrisināt variāciju optimizācijas problēmu. Kā aprakstīts (Kucukelbir, Ranganath et al., 2015), eksistē arī tā saucamās melnās kastes variāciju inferences metodes (Ranganath, Gerrish et al., 2013), kuras ļauj maksimizēt ELBO, taču bieži tās paļaujas uz stohastisko optimizāciju (Kucukelbir, Ranganath et al., 2015), tāpēc tās var uzskatīt uz apakšgrupu SVI algoritmiem.

Eksistē arī citas metodes Beiesa Variāciju Inferencei kuras īsi tiek aprakstītas šajā sekcijā. Strukturēta vidējā lauka variācijas inference (SMFVI) ir metode, kas balstīta uz MFV bet saglabā atkarības starp mainīgajiem (Hoffman & Blei, 2014). Šī metode ir praktiska, ja mainīgo attiecības ir zināmas. Neparametriskie modeļi (non-parametric), veic variāciju inferenci, izveidojot kompleksāku vērtību sadalījumu, kas var saturēt vairākas modas (T. V. Nguyen & Bonilla, 2013), ko uzskatāmi ilustrē attēls 1.2. ir apskatāms piemērs vairāku modu problēmai, kurai vienas modas varbūtību sadalījums nebūtu spējīgs pilnīgi raksturot īsto varbūtību.



(a)

1.2. att. Ilustrācija parāda iespējamu modeļa problēmu neidentificēt vairākas modas sadalījumā (Avci, Z. Li et al., 2021)

Tālāk tiek aprakstīts processs kā variāciju inferences algoritmi darbojas. Variāciju inference paļaujas uz Beiesa teorēmu, kurā gala varbūtības sadalījums tiek iegūts

no sākotnējās hipotēzes, ticamību hipotēzes notikšanai un patieso datu novērojumiem. Sākumā tiek definētas sākotnējais varbūtības sadalījums un iespējamības hipotēze (likelihood), tiek veikti novērojumi no pieejamās datu kopas un rezultātā, izmantojot Beiesa teoriju, tiek iegūta gala varbūtības sadalījums. Lai optimizētu modeli, tiek novērtēta rezultāta ELBO vērtīa, un tā tiek maksimizēta ar vienu no aprakstītajām metodēm.

$$ELBO = E[logP(X|z)] - KL[q(z|X)||p(z)] \tag{1.2}$$

kur, E - vidējā vērtība no novērojumiem;
$P(X|z)$ - nosacītā varbūtība punktam X, ņemot vērā latento mainīgo z;
KL - Kullback-Leibler divergence;
$q(z|X)$ - tuvinātā gala varbūtība;
$p(z)$ - sākotnējais varbūtības sadalījums latentajiem mainīgajiem z.

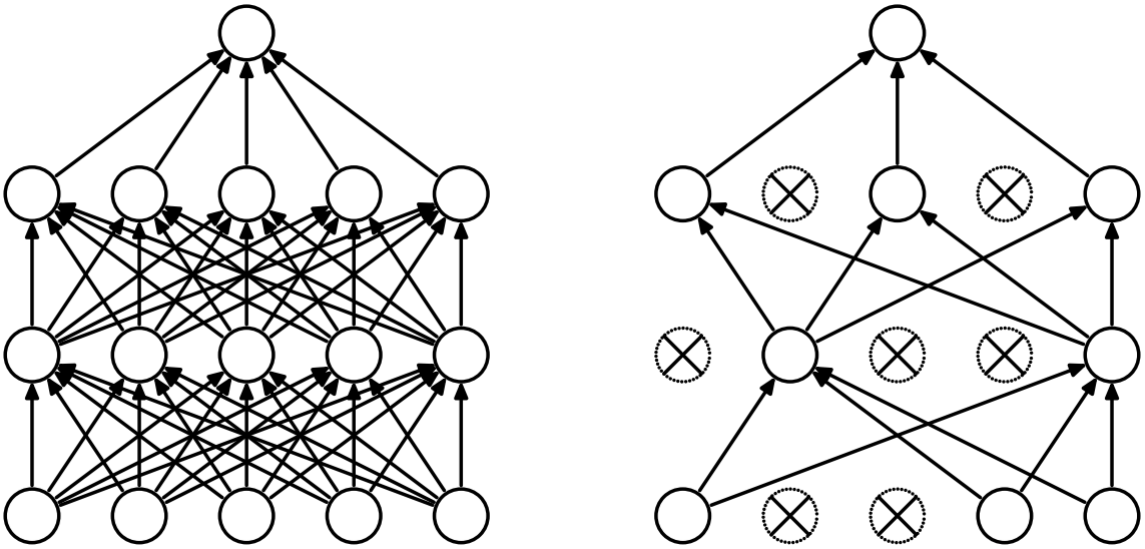
ELBO maksimizācija notiek mainot tuvinātās gala varbūtības parametrus - vidējo vērtību un dispersiju. Galā tiek iegūts gala varbūtības sadalījums ar vislielāko ELBO vērtību vai vismazāko KL novirzi, un modelis tiek uzskatīts par gatavu.

Ar rezultējošo varbūtību sadalījumu iespējams aprēķināt rezultātu jauniem ievaddatiem, izmantojot paraugu simulēšanu (Gelman, Carlin et al., 2013). Vērtību sadalījums tiek lietots, lai izveidotu prognozēto vērtību sadalījumu jauniem ievades datiem. Tad tiek aprēķināta vidējā vērtība un standartnovirze, lai noteiktu rezultāta nenoteiktību - standartnovirzes pieaugums norāda uz lielāku nenoteiktību.

1.2. Monte Karlo caurkrites metode

Monte Karlo caurkrite jeb MC-D (Monte Carlo Dropout) ir metode, kurā kāda daļa tīkla neironu nejaušā kārtībā tiek izslēgti, kā ilustrēts attēlā 1.3. MC-D ir plaši izplatīta metode tradicionālos neironu tīklos, ko izmanto, lai izvairītos no pārapmācības - tā samazina katra atsevišķā neirona ietekmi uz kopējo tīklu. Pēc modeļa apmācības caurkrite tiek izslēgta (Srivastava, Hinton et al., 2014). Šo metodi, kurā tiek izslēgti neironi var lietot arī citā viedā - lai aprēķinātu neironu tīkla ticamības mēru. Gal and Ghahramani (Gal & Ghahramani, 2015) parāda, ka, neizslēdzot caurkriti pēc apmācības, šo metodi var uzskatīt par Beiesa tuvināšanas metodi un iegūt vērtību sadalījumu, kas ir līdzvērtīgs Variāciju Inferences algoritmam. Monte Carlo metodes, izmantojot nejaušu neironu izslēgšanu (dropout), ļauj mums izveidot dinamisku modeli, kurš piedāvā atšķirīgus rezultātus katru reizi, kad tiek analizēts konkrēts datu punkts. Tas notiek tāpēc, ka izslēgtie neironi nesniedz ieguldījumu rezultāta veidošanā - tādējādi radot nepārtraukti mainīgu rezultātu modelim, kas tiek atkārtoti vērtēts. Monte Karlo caurkrites pieeja ir viegli implementējama jau iepriekš izstrādātiem mākslīgiem neironu tīkliem, un caur-

krites slāņi jau ir plaši izplatīti mūsdienu dziļās mašīnmācīšanās modeļos - tos pat var pielietot bez eksistējošo modeļu apmācības no jauna (Jospin, Buntine et al., 2020).



1.3. att. Pa kreisi redzams neironu tīkls ar visiem savienojumiem. Pa labi redzams neironu tīkls ar neironu caurkriti - neironi tiek izslēgti un neizmantoti izpildes laikā. (Srivastava, Hinton et al., 2014)

Lai izprastu šāda tipa modeļa raksturu, nepieciešams apskatīt, kā strādā tradicionāli mākslīgie neironu tīkli. Dziļie neironu tīkli sastāv no vairākiem neironu slāņiem, kuros katram neironam ir pievienots svars, kuram sākotnēji ir piešķirta nejauša vērtība, kā arī ir pievienots nobīdes koeficients. Ieejas dati tiek izplatīti tālākajos slāņos tīklā. Kļūdas atpakaļizplatīšana tīklā notiek no izejas slāņa uz visiem iepriekšējiem slāņiem līdz tiek sasniegti ieejas slāņi. Šajā procesā svaru un nobīdes koeficientu vērtības tiek koriģētas, ņemot vērā kļūdu. Optimizācija notiek pēc katras iterācijas, izmainot katra svara vērtību izmantojot kādu optimizācijas algoritmu (e.g.SGD, Adam utt.).

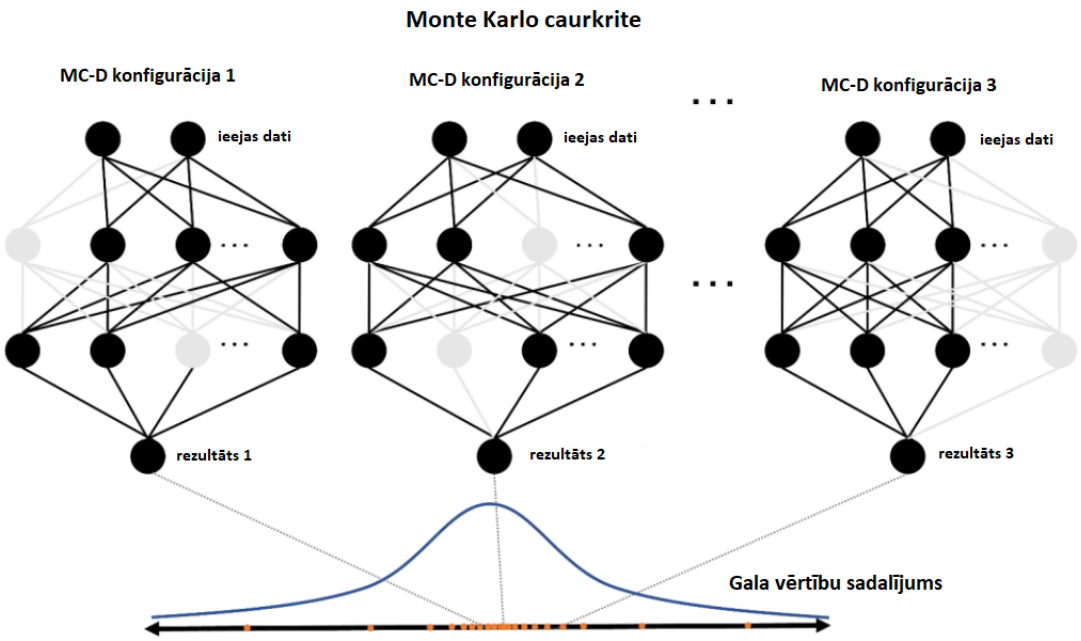
Monte Karlo tīklus ir iespējams strukturēt dažādos veidos, taču tie pirmatnēji tikai izstrādāti ar domu, ka atsevišķi neironi tiek izslēgti, balstoties uz kādu gadījuma faktoru. Taču tos var implementēt ar dažādiem slāņu skaitiem un aktivizācija funkcijām, kā rezultātā ir novērojami citi rezultāti (Srivastava, Hinton et al., 2014).

Pētījumi ir parādījuši, ka Monte Karlo caurkrites metode ir spējīga izveidot līdzīgus rezultātus variāciju inferences metodei (Gal & Ghahramani, 2015), kas ir izraisījis lielāku interese šāda tipa modeļos. Ir ievestas jaunas pieejas, kā izmantot Monte Karlo caurkriti kā inferences metodi, tā saucamās variāciju caurkrites metodes. Tās pārsvarā atšķiras ar to, kā caurkrites koeficients tiek noteikts un ieviests katrā tīkla neironā. Metodes kā strukturētā variāciju caurkrite (Structured variational) (S. M. Nguyen, D. Nguyen et al., 2021), Gausa caurkrite (Gaussian dropout) vai tā sauktā ”Concrete dropout” (Gal, Hron et al., 2017) cenšas izmantot apmācāmu varbūtību sadalījumu caurkrites koefi-

cientam, tādā veidā ļaujot caurkrites koeficientam būt adaptīvam datu kopai Kingma, Salimans et al., 2015. Līdzīgi, tā saukta ”Sparse VD” metode cenšas ieviest caurkrites parametru katram neironam (Molchanov, Ashukha et al., 2017). Citi pētījumi ir apskatījuši metodes, kā caurkrites metodi pielietot ne tikai atsevišķiem neironiem vai to svariem, bet grupām neironu dažādās dimensijās. Šādas metodes ir parādijušas, ka ir ļoti vērtīgas, lai uzlabotu modeļu noturīgumu (Lin, P. Liu et al., 2019). Pētījumā (Kendall & Gal, 2017), autori parāda, kā modelēt aleatorisko un epistēmisko nenoteiktību izmantojot MC-D algoritmu gan regresijas, gan klasifikācijas modeļiem.

Šajā sekcijā tiek sniegts detalizēts ieskats Monte Karlo caurkrites algoritmu darbības procesā, kura galvenais mērķis ir izveidot vērtību sadalījumu. Tas sākas ar neironu tīkla modeļa izveidi, kas kalpo kā bāzes struktūra turpmākajam algoritmu darbības procesam. Neironu tīkla modeļa izveidei seko caurkrites realizācija. Šajā solī var tikt izmantotas divas galvenās metodes: caurkrites slāņu pievienošana vai variācijas caurkrites metodes pielietošana. Šo metožu pielietošana var notikt atsevišķiem neironiem vai arī neironu grupām. Caurkrites metožu ieviešana nodrošina noteiktu elastīgumu un mainīgumu modeļa darbībā, kas veicina dažādu rezultātu iegūšanu nākamajos procesa posmos.

Caurkrites implementācijai seko modeļa apmācība. Šajā procesā tiek izmantota daļa no kopējā datu kopuma, savukārt modeļa optimizācijai tiek izmantota kāda no optimizācijas metodēm. Kad modelis ir pilnībā apmācīts, tas tiek piemērots katram datu ierakstam testa datu kopā - modelis tiek izpildīts vairākas reizes uz katru datu ierakstu, un katrā izpildē tiek iegūts nedaudz atšķirīgs rezultāts. Atšķirīgie rezultāti, kas iegūti izpildot modeli uz datu ierakstiem, tiek apvienoti vērtību sadalījumā. Šāds vērtību sadalījums ir redzams attēlā 1.4. Katrs iegūtais rezultāts sniedz unikālu vērtību, kas kopā veido plašu potenciālo iznākumu spektru, kas iegūts izmantojot Monte Karlo caurkrites algoritmu. Šis procesa apraksts ir būtisks, lai labāk saprastu, kā tiek veidotas vērtību sadalījums MC-D algoritmam.



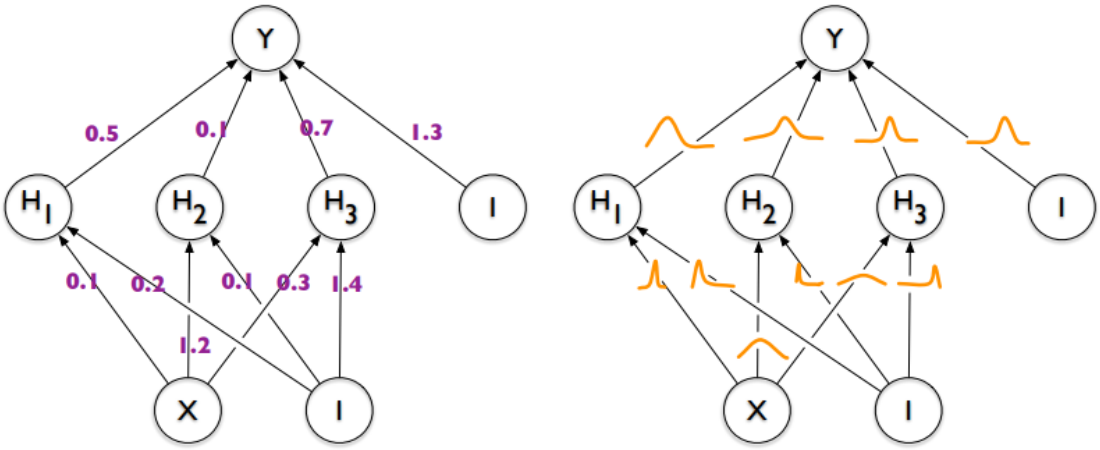
1.4. att. Ilustratīvs atspoguļojums kā tiek iegūts vērtību sadalījums MCD metodē (Avci, Z. Li et al., 2021)

1.3. Beies caur atpakaļizplatīšanos metode

Atpakaļizplatīšanās tradicionāli ir mehānisms, kas ļauj aprēķināt gradientu kļūdas funkcijai katram neironu tīkla neironam, tā palīdzot pieskaņot katra neirona svaru nākamajai apmācības iterācijai (Nielsen, 2015). Šis mehānisms ir ļoti plaši pielietots un efektīvs veids, kas ļauj neironu tīkliem tikt apmācītiem, tāpēc tā pielietošana arī Beiesa ietvarā ir saprotama.

Beiesa caur atpakaļizplatīšanos jeb BBB (Bayes by backprop) ir algoritms, kas tika ieviests pētniecības darbā "Weight Uncertainty in Neural Networks" (Blundell, Cornebise et al., 2015). Tajā viņi paskaidro, ka algoritms sastāv no neironu tīkla, kuros izmantoti variāciju inferences principi. Būtībā, BBB ir algoritms, kas veic neironu tīkla svaru izvērtēšanu, izmantojot vērtību sadalījumus, nevis fiksētas vērtības. Konceptuāla vizualizācija šim procesam ir redzama attēlā 1.5. Šīs pieejas mērķis ir ieviest nenoteiktību svaru vērtībās, lai novērtētu pārliecinātību modeļa prognozēšanas precizitātē. Šāda apmācība tiek veikta, maksimizējot variāciju izmaksu funkciju ELBO. Algoritma optimizācijai tiek pielietota gan kļūdas funkciju, gan KL novirze. BBB autori apgalvo, balstoties uz saviem pētījumiem, ka šī algoritma darbība ir pielīdzināma MC-D algoritmam savā veiktspējā.

BBB izmanto reparametrizācijas trika metodi, lai novērtētu un optimizētu svaru sadalījumu parametrus neironu tīklā. Mainīgo reparametrizācija jau ilgu laiku ir izmantota metode statistikas literatūrā, taču tikai nesen tai ir atrasti pielietojumi uz gradientu balstītā mašīnapmācībā (Kingma & Welling, 2013). Reparametrizācijas trika metode



1.5. att. Pa labi novērojamas punkta vērtības katram svaram kā tradicionālā neironu tīklā. Pa kreisi var redzēt vērtību sadalījumus, kas tiek pievienoti katram svaram, kā tas notiek Beiesa caur atapakaļizplatīšanos algoritmam (Blundell, Cornebise et al., 2015)

ir noderīga, kad mēs mācāmies parametrus, kas definē vērtību sadalījumu, piemēram, neironu tīkla svaru sadalījumam.

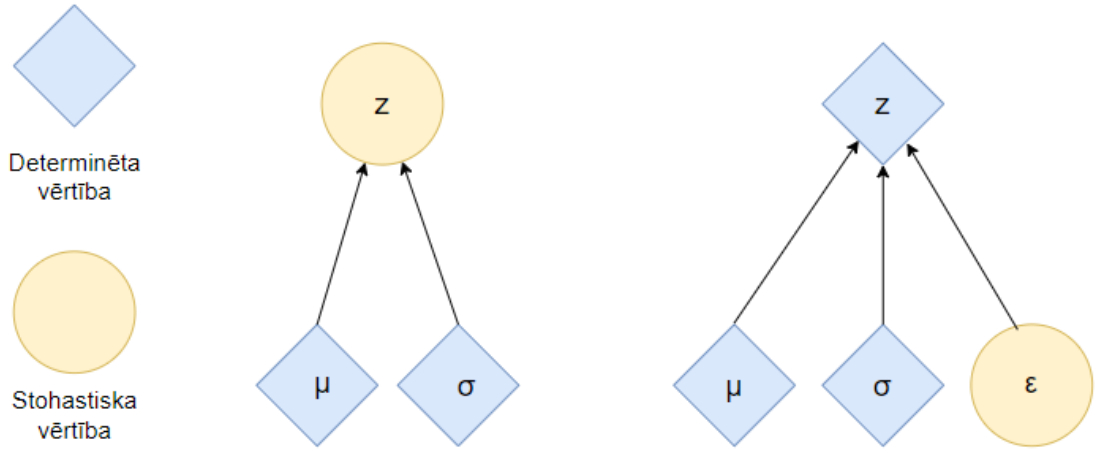
BBB algoritma kontekstā, neironu tīkla svaru sadalījums tiek reprezentēts kā normālsadalījums ar apmācāmiem parametriem μ un σ. Neironu tīkli atpakļizplatīšanās laikā izmanto gradientu nolaišanās metodi. Lai to izdarītu, mums jāaprēķina tīkla kļūdas funkcijas gradienti attiecībā uz tīkla svariem. Šie gradienti mums pasaka, kā mēs varam mainīt tīkla svarus, lai samazinātu tīkla kļūdu. BBB algoritmam mums ne tikai jāzina, kā kļūda mainās attiecībā uz tīkla svariem, bet mums arī jāzina, kā tā mainās attiecībā uz svaru sadalījuma parametriem, kas izveido problēmu, jo gradienta aprēķināšana stohastiskām vērtībām var būt ļoti sarežģīta un veidot kļūdas to piemītošās nejaušības dēļ. Reparametrizācijas trika metode ļauj apiet šo problēmu. Tā vietā, lai mēģinātu aprēķināt gradientus attiecībā uz stohastiskā svara sadalījuma parametriem, mēs varētu pārveidot problēmu tā, lai būtu jāaprēķina gradienti attiecībā uz determinētu svara sadalījumu. Mēs to veicam, pārveidojot paraugus no standarta normālsadalījuma, izmantojot mūsu svara sadalījuma parametrus. Šī metode tiek realizēta sekojoši: vispirms tiek iegūts paraugs ϵ no standarta normālsadalījuma $N(0, I)$, un pēc tam šis paraugs tiek pārveidots, izmantojot mūsu svara sadalījuma parametrus μ un σ. Šī pārveidošana tiek aprakstīta ar formulu (1.3).

Šeit $g_\varphi(\epsilon, x)$ ir vektora vērtības funkcija, ko parametrizē $\varphi = \mu, \sigma$, un ϵ ir paraugs no standarta normālsadalījuma. Iegūtais z ir paraugs no mūsu svara sadalījuma, un tā kā šī pārveidošanas funkcija ir determinēta un diferencējama, tad gradientu aprēķināšana ir vienkārša, izmantojot standarta atpakaļizplatīšanas metodi. BBB arī var nosaukt par praktisku implementāciju SVI algoritmam ar reparametrizācijas triku (Jospin, Buntine

et al., 2020). Attēlā 1.6. iespējams apskatīt vizuālu attēlojumu reparametrizācijas trika metodei.

$$z = g_{\varphi}(\epsilon, x) = \mu + \sigma * \epsilon \tag{1.3}$$

kur, z mainīgais, ko mēs aprēķinām;
g_{φ}) - funkcija, kas aprēķina z vērtību, izmantojot ϵ un x. φ šeit apzīmē funkcijas parametrus;
ϵ - gadījuma mainīgais, kas parasti seko normālajam sadalījumam ar vidējo vērtību 0 un standarta novirzi 1;
μ - vidējā vērtība jeb centra punkts sadalījumā;
$sigma ir standarta novirze, kas mēra sadalījuma platumu.



1.6. att. Ilustrācija atspoguļo reparametrizācijas trika metodi - kreisajā pusē redzams, ka izvades vērība ir stohastiska, labajā pusē redzams, ka stohastiskais parametrs ϵ tiek parametrizēts iegūstot determinētu vērtību z.

Sākumā iek izvēlēts sākotnējais vērtību sadalījums. Katram datu punktam apmācības datu kopā tiek paņemts paraugs no sākotnējais varbūtības sadalījums, datu punkti tiek izplatīti caur tīkla slāņiem un iegūtajam vērtību sadalījumam aprēķināta ELBO vērtība. Šī vērtība tiek optimizēta atpakaļizplatīšanās laikā, izmantojot stohastisko gradientu nosēšanos (SGD). Šis process tiek atkārtots līdz iestājas konverģence.

1.4. Citas Beiesa neironu tīklu metodes

Eksistē citi algoritmi, kurus arī iekļauj Beiesa neironu tīklu kategorijā, šajā sadaļā tie tiek īsumā aprakstīti.

Markova Ķēžu Monte Karlo jeb MCMC (Markov Chain Monte Carlo) ir plaša algoritmu klase, ko izmanto, lai veidotu izlases no varbūtību sadalījumiem (Jespersen,

2010). Šīs metodes implementācija sākas izvēloties kādu nejaušu mainīgo x un izveidojot Markova ķēdi stāvokļa telpā S. Izveidotajai Markova ķēdei ir specifisks stacionārs sadalījums, kas nozīmē ka Markova ķēdes pāreju sadalījums eventuāli konverģēs uz mērķa sadalījumu. Mērķa sadalījums ir šis stacionārais sadalījums. Pēc pietiekami daudz iterācijām, kad stāvokļa sadalījums uz Markova ķēdes ir pietiekami tuvu specifiskajam stacionārajam sadalījumam, no mērķa sadalījuma var tuvināt izlases vērtību ar Markova ķēdes simulāciju (Y. Zhang, X. Li et al., 2020). Hamiltona Monte Karlo un Gibsa izlašu veidošana ir MCMC algoritmu piemēri, kas bieži tiek izmantoti Beiesa neironu tīklos.

Pētījumā (Salimans, Kingma et al., 2014), tiek parādīts, kā apvienojot MCMC ar VI metodēm var gūt precīzākus vērtību tuvinājuma rezultātus kā izmantojot tikai vienu metodi atsevišķi.

Varbūtiskā atpakaļizplatīšana, kas aprakstīta (Hernández-Lobato & Adams, 2015), ir metode, kurā neironu svari netiek definēti kā punkta vērtības, bet gan kā grupa ar viendimensionālām Gausa normālsadalījuma funkcijām, katra tuvinājuma galējo vērtību sadalījumu. Līdzīgi kā tradicionālos neironu tīklos, ievades dati tiek izplatīti caur tīklu, taču tāpēc, ka neironu svari ir nejauši, aktivācijas kas tiek izveidotas arī ir nejaušas katrā slānī un rezultējas sadalījumos. Secīgi tiek tuvināts katrs sadalījums ar grupu gausa normālsadalījumu funkcijām, kas sakrīt ar to vidējām vērtībām un dispersijām. Tālāk algoritms aprēķina marginālās varbūtības logaritmu mērķa mainīgajam. Atpakaļizplatīšanās šajā algoritmā notiek, aprēķinot gradientu šo sadalījumu vidējai vērtībai un dispersijai.

Bez iepriekš pieminētajām, eksistē citas metodes, kas modelē pārliecības mēru, taču šajā darbā netiek apskatītas: Beiesa aktīvās apmācības jeb Bayesian Active Learning (BAL), Variational Autoencoders, Lapasa tuvinājumi u.c. Par tām aprakstīts literatūras avotā (Abdar, Pourpanah et al., 2021).

1.5. Salīdzināšanas metodes

Šajā darbā cenšos salīdzināt mašīnmācīšanās metodes varbūtības sadalījuma izveidošanai. Lai to panāktu ir nepieciešams apskatīt, kā ir iespējams salīdzināt dažādus varbūtības sadalījumus. Atkarībā no tā, vai uzdevums, kas tiek veikts, ir regresijas uzdevums vai klasifikācijas uzdevums ir iespējams apskatīt dažādas pieejas. Bieži regresijas algoritmu salīdzināšanai tiek salīdzinātas kļūdas vērtības, tas būtu iespējams šajā gadījumā, izmantojot sadalījuma viduspunktus. Līdzīgi, klasifikācijas uzdevumos ir iespējams novērtēt modeļu pareizi klasificētos rezultātus un salīdzināt modeļu procentuāli pareizi klasificētos rezultātus.

Regresiju uzdevumos, kuros mēs mēģinām paredzēt kvantitatīvus mērķa mainīgos, ir daudz dažādu metožu modeļa veiktspējas novērtēšanai. Bieži izmantotas meto-

des ietver vidējo absolūto kļūdu, zināmu arī kā Mean Absolute Error (MAE) ar formulu (1.5), vidējo kvadrātisko kļūdu (MSE) ar formulu (1.6), Hūbera kļūdas (Huber loss) funkciju (L_δ) ar formulu (1.4) un vidējās kvadrātisko kļūdu (RMSE) (Jospin, Buntine et al., 2020).

$$L_\delta = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & if\ |(y-\hat{y})| < \delta \\ \delta((y-\hat{y}) - \frac{1}{2}\delta) & citdi \end{cases} \tag{1.4}$$

kur, L_δ - Hūbera kļūda;
y - patiesā vērtība, pret kuru tiek salīdzinātas prognozes;
$\hat{y}$ - prognozētā vērtība;
δ - slieksnis, kas nosaka, kura no divām zaudējumu funkcijām tiek izmantota.

$$MAE = \sum_{i=1}^{N} |x_i - y_i| \tag{1.5}$$

kur, MAE - vidējā kvadrātiska kļūda;
N - kopējais novērojumu skaits;
x_i - prognozētā vērtība i-tajam novērojumam;
y_i - patiesā vērtība i-tajam novērojumam.

$$MSE = \sum_{i=1}^{D} (x_i - y_i)^2 \tag{1.6}$$

kur, MSE - vidējā kvadrātiska kļūda;
D - kopējais novērojumu skaits;
x_i - prognozētā vērtība i-tajam novērojumam;
y_i - patiesā vērtība i-tajam novērojumam.

Citas formulas ļauj novērtēt apjomu starp vidējo vērtību un paredzētajiem (prognozētajam) intervāliem. Prognozes intervāla seguma varbūtība (1.7) jeb Prediction interval coverage probability (PICP) (González-Sopeña, Pakrashi et al., 2021).

$$PICP = \frac{1}{N}\sum_{i=1}^{N} c_i, \text{ kur } c_i = \begin{cases} 1, & if y_i \in {}^{\alpha}_{i} \\ 0, & otherwise \end{cases} \tag{1.7}$$

kur, $PINC$ - prognozes intervāla seguma varbūtība;
N - kopējais novērojumu skait;
c_i - bināra vērtība, kas apraksta, vai patiesā vērtība y_i iekrīt prognozētajā intervālā ${}^{i\alpha}_{i}$;
y_i - patiesā vērtība vai etiķete, kas atbilst i-tajam novērojumam.;
α_i - prognozētais intervāls i-tajam novērojumam.

Pētījumā (Gal & Ghahramani, 2015) autori salīdzina RMSE un standartnovirzes

vērtības 10 dažādām datu kopām un salīdzina BI, PBP un MC-D algoritmus, kopumā apskatot viņu sniegtos rezultātus, varam novērot, ka MC-D izveido labākus rezultātus nekā VI metode un, izņemot atsevišķām datu kopām, labāk arī par PBP implementāciju. Pētījumā (Y. Zhang, X. Li et al., 2020) autori izmanto RMSE un MAPE, lai novērtētu novirzes apjomu starp vidējo vērtību paredzētajam intervālam, kā arī PICP un PINAW (Prediction interval normalized average width) metrikas, lai novērtētu paredzētos intervālus starp modeļiem, galā secinot, ka viņu implementētā MC-D metode ir pārāka par MCMC metodi. Citā pētījumā (Olivier, Shields et al., 2021), tika uzkonstruētas trokšņainas polinoma un sīnusa funkcijas un, izmantojot HMC metodi tika uzģenerēts mērķa vērtību sadalījums ar augstāku veiktspēju nekā BBB un VI metodes. Šajā pētījumā autori uzsver, ka sākuma sadalījumi un modeļu hiperparametru izvēle spēcīgi ietekmē modeļu veiktspēju gan vidējo vērtību, gan nenoteiktības mērā. Autori secina, ka apvienojot dažādus VI modeļus ar atšķirīgiem hiperparametriem, tiek iegūti uzlaboti vidējo vērtību un nenoteiktības rezultāti. Pētījumā (Blundell, Cornebise et al., 2015), kura autori iepazīstina BBB metodi, to salīdzina ar MC-D un tradicionālu neironu tīklu. Pirmajā eksperimentā, salīdzinot klasifikācijas problēmas kļūdu skaitu, kurā tiek secināts, ka BBB ir spējīgs prognozēt ar vismazāko kļūdu. Kā arī otrajā eksperimentā, izmantojot tā saukto koteksta bandītu (contextual bandits) problēmu, parāda, ka BBB algoritms ir spējīgs veikt efektīvu izpētes un izmantošanas kompromisu, kas var būt nepieciešams stimulētā mašīnmācīšanās uzdevumā.

Šķērsentropija (Cross-entropy) (CE) ar formulu (1.8) ir plaši izplatīta kļūdas funkcija, ko pielieto klasifikācijas algoritmu optimizācijai. To definē kā mēru, kas nosaka, cik paredzētā varbūtība atšķiras no patiesās.

$$CE = -(y \log(p) + (1 - y) \log(1 - p)) \tag{1.8}$$

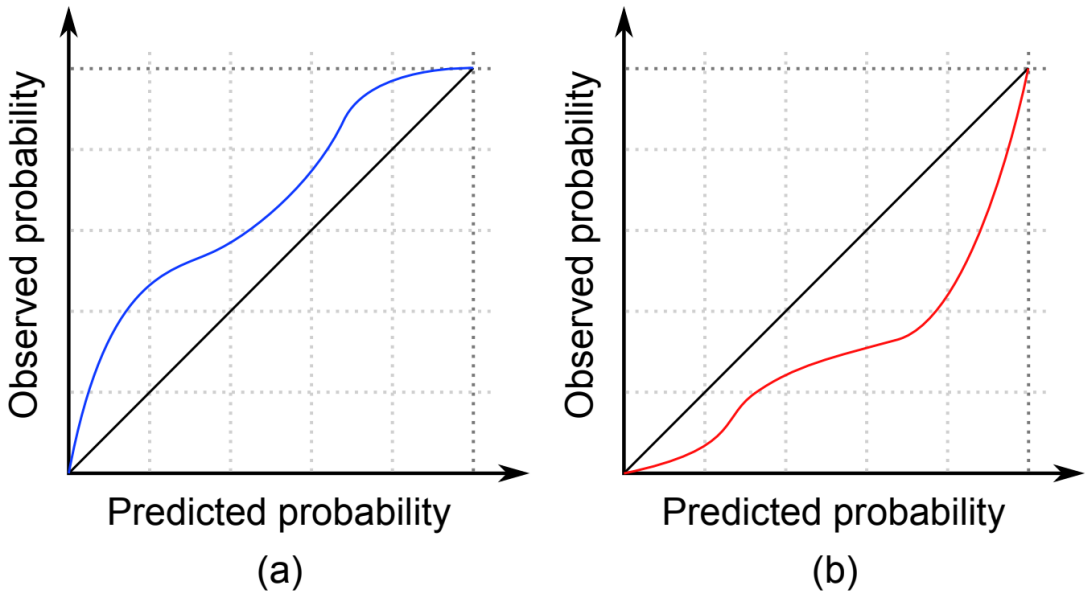
kur CE - Šķērsentropijas kļūdas funkcija;
y - patiesa klasificējamā ieraksta vērtība (1 vai 0);
p - varbūtība, ka rezultāts ir 1.

Jo mazāka ir šķērsentropijas vērtība, jo precīzāka ir modeļa klasificēšana. Šo kļūdas mēru var efektīvi izmantot atpakaļizplatīšanās algoritmā, lai veiktu modeļa svaru uzlabojumus.

Kā aprakstīts (Jospin, Buntine et al., 2020), pieņemta metode modeļa kalibrācijas noteikšanai ir uzticamības diagramma. Tā tiek ilustrēta attēlā 1.7. - šajā diagrammā, tiek atspoguļota, paredzētā varbūtība ar novēroto varbūtību. Katras klases varbūtības diagramma var atsevišķi tikt salīdzināta ar pārējām klasēm. Piemēru tā izmantošanā iespējams atrast (Kendall & Gal, 2017).

Līdzīga metrika modeļa kalibrācijas noteikšanai ir sagaidāmās kalibrācijas kļūda (Expected Calibration Error). Šī vērtība raksturo katra modeļa izvadītā vērtību intervāla

pareizi identificēto vērtību īpatsvaru (Nixon, Dusenberry et al., 2019).



1.7. att. Uzticamības diagramma piemeri nepietiekami un pārlieku pārliecinātiem modeļiem (Jospin, Buntine et al., 2020)

Kļūdu matrica ir viens no biežāk lietotajiem instrumentiem, lai novērtētu klasifikācijas algoritmus, un tas sniedz ieskatu par klasifikācijas modeļa veiktspēju. Kļūdu matrica ir kvadrātveida matrica, kas ietver informāciju par modeļa pareizu un nepareizu klasifikāciju. Tā satur četrus galvenos elementus - pareizi pozitīvo (TP), kļūdaini pozitīvo (FP), kļūdaini negatīvo (FN) un pareizi negatīvo (TN), kas atspoguļo modeļa prognozēto un faktisko klasi. TP un TN attēlo gadījumus, kad modelis pareizi prognozē klasi, bet FP un FN attēlo situācijas, kad modelis kļūdās. Sēņu datu klasifikācija ir labs piemērs, kas demonstrē, kāpēc ir svarīgi analizēt kļūdu matricu - jādomā, kas ir bīstamāk, nepatiesi identificēt sēni kā ēdamu vai kā indīgu. Pašsaprotami, ka nepareizi identificējot sēni kā ēdamu var rasties negatīvākas sekas, jo cilvēks var, paļaujoties aprakstītai identifikācijai, pakļaut sevi veselības riskam.

Balstoties uz (Tötsch & Hoffmann, 2020), precizitāte ir metrika klasifikācijas uzdevumu novērtēšanai. Precizitāte ir daļa no kopējiem klasifikācijas rezultātiem, kuri tika klasificēti pareizi. Precizitāti aprēķina pēc formulas (1.9):

$$\text{Precizitāte} = \frac{TP + TN}{TP + FN + FP + TN} \tag{1.9}$$

kur, TP – pareizi pozitīvi klasificēto vērtību skaits;
TN – pareizi negatīvi klasificēto vērtību skaits;
FN - kļūdaini negatīvi klasificēto vērtību skaits;
FP - pareizi negatīvi klasificēto vērtību skaits.

Algoritmu apmācības procesā, MC-D un BBB algoritmiem ir iespējams sekot precizitātes mērījumiem un standartnovirzes vērtībām, izmantojot grafikus, kas attēlo abu algoritmu veiktspēju dziļās mācīšanās procesā. Grafikos tiks attēlotas abu algoritmu precizitātes vērtības un standartnovirzes vērtības, kas tiks novērtētas katrā apmācības ciklā, analizējot visus datu partiju vidējos precizitātes rezultātus.

Pārklājums (Recall) ir metrika, kas nosaka pozitīvi novērtēto gadījumu skaitu pret kopējo pozitīvo gadījumu daudzumu. To apraksta ar funkciju (1.7) Šī metrika nosaka, cik algoritms spēj pareizi identificēt pozitīvos gadījumus, ignorējot nepareizi identificēto negatīvo gadījumu skaitu.

$$\text{Pārklājums} = \frac{TP}{TP + FN} \tag{1.10}$$

kur, TP – pareizi pozitīvi klasificēto vērtību skaits;
FN - kļūdaini negatīvi klasificēto vērtību skaits.

F1 funkcija (1.8) ir cita metrika, kas tiek izmantota klasifikācijas algoritmu novērtēšanai. Tā tiek definēta kā harmoniskā vidējā vērtība no precizitātes un pārklājuma. To aprēķina izmantojot gan precizitātes, gan pārklājuma vērtības.

$$F1 = 2 * \frac{\text{Precizitāte} * \text{Pārklājums}}{\text{Precizitāte} + \text{Pārklājums}} \tag{1.11}$$

kur, F1 – F1 funkcijas vērtība.

Dispersija un standartnovirze ir rādītāji, kas nosaka izlases variāciju jeb izkliedi ap izlases vidējo vērtību. Šis rādītājs var tikt izmantots, lai veiktu secinājumus par algoritma pārliecinātību. Dispersiju apzīmē ar σ^2 un aprēķina ar formulu (1.12). Standartnovirzi apzīmē ar σ un tās formula (1.13) ir kvadrātsakne no dispersijas. Lielāka dispersija nozīmē mazāku parliecinātību rezultātos, un mazāka dispersija nozīmē lielāku pārliecinātību - tāpēc, ka algoritma izkliedei esot augstai, tas nav vienmērīgi novērtējis gala vērtības rezultātus.

$$S^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n - 1} \tag{1.12}$$

kur, S^2 - dispersija;
x_i - i-tais novērojums;
$\bar{x}$ - parauga vidējā vērtība;
n - ir paraugu skaits.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{1.13}$$

kur, σ - standarta novirze;
x_i - i-tā novērojuma vērtība;
$\bar{x}$ - vidējā vērtība visiem novērojumiem;
n - ir novērojumu skaits.

Lai novērtētu prognožu pārliecības mēru klasifikācijas uzdevumiem var tikt izmantota entropija. Entropija informācijas teorijas kontekstā raksturo gadījumlieluma nenoteiktību. To var aprēķināt pēc formulas (1.14). Mazāka entropija nozīmē efektīvāku algoritmu un precīzāku klases atdalīšanu. Šī metrika ļauj arī saprast, cik labi algoritms atdala dažādas kategorijas un tāpēc var palīdzēt identificēt algoritmu, kuru pielietot lielākas vai mazākas datu pārklāšanās gadījumos.

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{1.14}$$

kur, $H(X)$ - entropija;
p_i - i-tā notikuma varbūtība;
$\log_2$ - logaritms ar bāzi 2, kas tiek izmantots informācijas teorijā;
n - notikumu skaits.

Marķējumu trokšņa injekcija (label noise injection), ir metode, kurā klasifikācijas apmācības procesā, tiek apzināti izmainītas, tādā veidā ieviešot troksni datu kopā. Šai pieejai ir daudz asociētu pētījumu, kas apraksta gan tās praktisko pielietojamību, gan šīs pieejas izmantošanu eksperimentos, lai novērtētu kādu neironu tīkla īpašību. Ir novērots, ka trokšņa injekcijas var būt noderīga, lai palielinātu neironu tīklu noturīgumu (Reed, H. Lee et al., 2014). Citi pētījumi apskata risinājumus, kas mēģina attīrīt datus, lai uzlabotu klasifikācijas algoritmu novērtēšanas spēju (Chen, Yu et al., 2021). Atkal citi mēģina apskatīt pieejas, kā uzlabot zuduma funkcijas noturīgumu trokšņainiem datiem (Zhou, X. Liu et al., 2021). Pētījums (Shridhar, Laumann et al., 2018) apskata, ko rada trokšņa pievienošana, mainot nejauši izvēlētus pikseļus attēlos - šis darbs demonstrē, ka, neskatoties uz trokšņa klātbūtni, aleatoriskā nenoteiktība paliek nemainīga.

Apzināta trokšņa injekcija var tikt izmantota, lai novērtētu dažādas tehnikas trokšņainu datu apstrādei klasifikācijas uzdevumos (J. Li, Socher et al., 2020). Ir pētījumi, kas salīdzini dažādās pieejas trokšņainu datu kopu apmācībai (Reed, H. Lee et al., 2014). Mākslīgo tīklu algoritmi ir spējīgi modelēt troksni datu kopā, jo to kļūda var samazināties līdz nullei apmācības laikā, kad apmācība izmantoti nejauši izmainītas rezultātu kategorijas. Taču šī trokšņa ieviešana samazina tīkla precizitāti testēšanas datiem, jo

tiem nav vienādas attiecības ar trokšņainiem datiem, kas izmantoti apmācībā (C. Zhang, Bengio et al., 2016).

Trokšņa ieviešana var būt vērtīga mašīnmācīšanās algoritmu novērtēšanai un salīdzināšanai, jo varam salīdzināt, kā algoritms reaģētu dažādu datu kvalitātes klasifikāciju. Šāda salīdzināšanas pieeja jau ir izmantota pētījumā, kurā tika salīdzinātas dažādas apmācības pieejas vizuālām klasifikācijas problēmām, izmantojot bilžu datu kopas CIFAR-10, CIFAR-100 un SVHN (K. Lee, Yun et al., 2019).

Šajā pētījumā tiek analizēts, cik labi dažādi algoritmi modelē vērtību sadalījumu un rezultējošo noturīgumu dažādiem datu kvalitātes stāvokļiem. Lai to veiktu, šājā darbā tiks izmantota marķējumu jaukšana ievaddatem un izvaddatiem no apmācāmās datu kopas. Mērķis ir apskatīt, kā prognožu rezultāti mainās, palielinoties troksnim apmācāmajā datu kopā.

2. METODOLOĢIJA

Šī darba mērķi ir salīdzināt algoritmu stabilitāti un to izveidoto dispersiju. Tas tiek panākts, izmantojot klasifikācijas uzdevumu. Katrs no algoritmiem tiek izmantots, lai no testu kopas ievaddatiem paredzētu klasi - tas tiek darīts dažādos datu trokšņu apstākļos. Salīdzinājumi eksperimentiem ir pamatoti un izpildīti empīriski. Algoritmi savā starpā atšķiras, bet datu kopas sagatavošana ir vienāda starp algoritmiem. Šajā darba daļā, tiek detalizēti aprakstīta metodoloģija, ar kuras palīdzību tiek īstenoti eksperimenti.

2.1. Datu kopa

Šī darba veikšanai tiek lietota sēņu datu kopa, kas satur 8124 ierakstus un 23 dažādas iezīmes par dažādām sēnēm, kas redzamas tabulā 2.1. Šo datu kopu ir iespējams izmantot, lai klasificētu sēnes indīgajās un ēdamās. Datu kopa satur 22 raksturojošas iezīmes un 1 klasifikācijas kolonu - indīga (p) vai ēdama (e). Šajā datu kopā visi dati, izņemot vienu iezīmi, ir aizpildīti un nav trūkstošu datu. Iezīmei ”stalk root” nav pievienotas 2480 vērtības.

Novērots, ka ēdamo sēņu skaits datu kopā ir 3916, bet indīgo - 4208, tātad datu kopa ir samērīgi balansēta. Šajā datu kopā nav neviena skalāra parametra, tikai kategoriskas vērtības. Kategorijas katram parametram ir vairākas, izņemot tā saukto sēnes plīvura iezīmi ”veil-type”, kuram visas sēnes atbilst vērtībai p. Citi parametri ir kategorizēti no 2 līdz 12 dažādās kategorijās, kuras var apskatīt tabulā 2.1. Lai apskatītu iezīmes vel detalizētāk, tiek vizualizēti iezīmju iespējamo kategoriju vērtības katrai no klasēm - indīga vai ēdama, šīs vizualizācijas novērojamas attēlā 2.1.

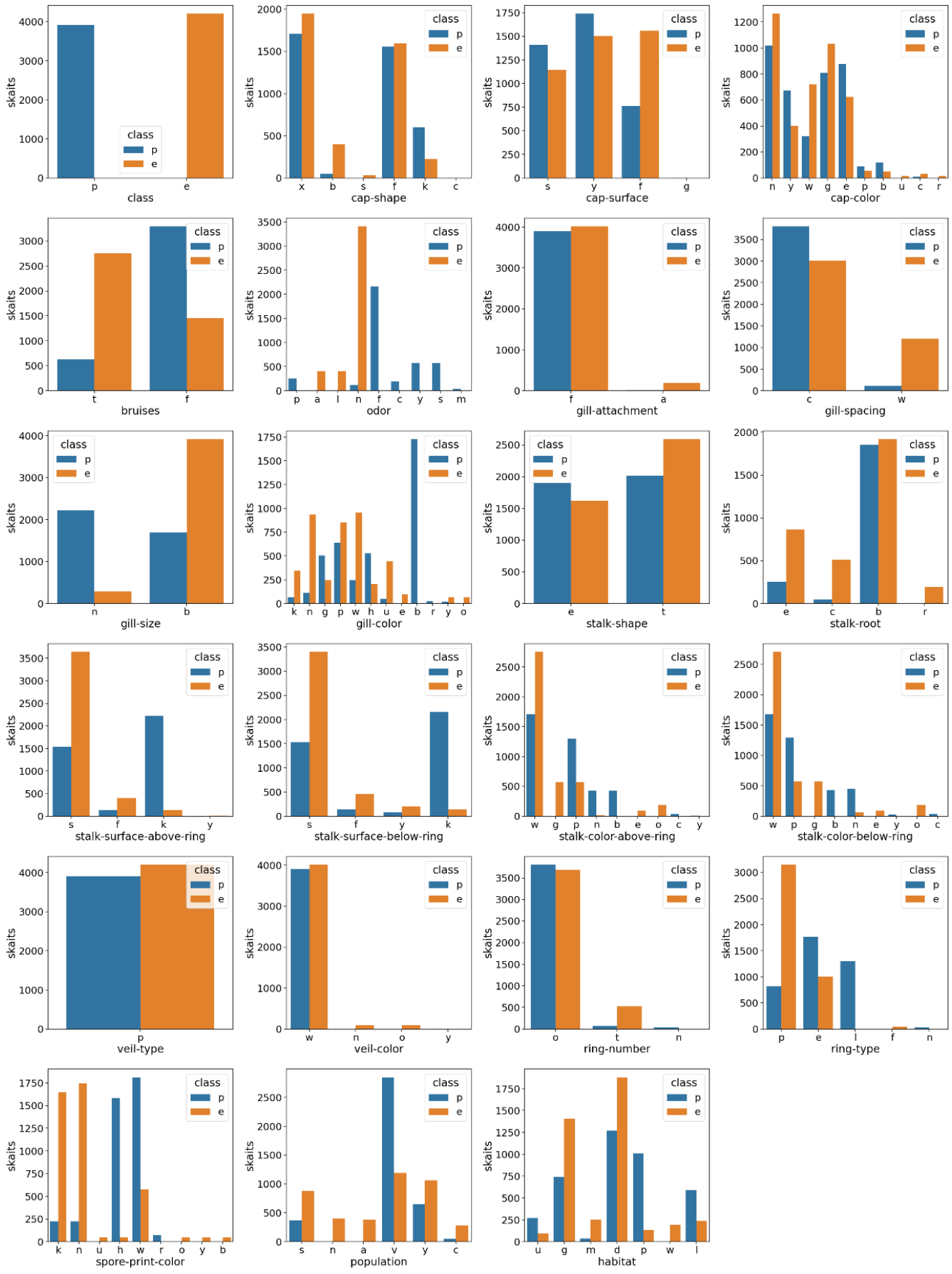
Analizējot šo sēņu datu kopu tiek veikti sekojošie novērojumi (Val, 2021):

1. Ēdamām sēnēm biežāk ir ar nobrāzumiem (bruising)
2. Ēdamām sēnēm biežāk nav smarža, bet indīgām sēnēm ir slikta smarža (odor)
3. Ēdamām sēnēm kāti ir gludi (smooth), bet indīgām tie bija zīdaini (silky)
4. Lapiņu ēdamām sēnēm ir lielas un platas, bet indīgām tās ir gan lielas, gan šauras
5. Sporu nospiedumu krāsa ēdamām sēnēm parasti ir melna vai brūna, indīgām sēnēm tā ir šokolādes krāsā vai balta.

Šo secinājumu autors apgalvo ka ir sasniedzis 100% precizitāti savam klasifikācijas algoritmam, ko viņš veica ar loģistiskās regresijas algoritma palīdzību. Tas liek sagaidīt arī augstu precizitāti šajā darbā izmantotajiem algoritmiem.

Iezīme	Iespējamās vērtības	Dažādo vērtību skaits	Tukšo vērtību skaits
cap-shape	b, c, x, f, k, s	6	0
cap-surface	f, g, y, s	4	0
cap-color	n, b, c, g, r, p, u, e, w, y	10	0
bruises	t, f	2	0
odor	a, l, c, y, f, m, n, p, s	9	0
gill-attachment	a, d, f, n	4	0
gill-spacing	c, w, d	3	0
gill-size	b, n	2	0
gill-color	k, n, b, h, g, r, o, p, u, e, w, y	12	0
stalk-shape	e, t	2	0
stalk-root	b, c, u, e, z, r, ?	7	2480
stalk-surface-above-ring	f, y, k, s	4	0
stalk-surface-below-ring	f, y, k, s	4	0
stalk-color-above-ring	n, b, c, g, o, p, e, w, y	9	0
stalk-color-below-ring	n, b, c, g, o, p, e, w, y	9	0
veil-type	p	1	0
veil-color	n, o, w, y	4	0
ring-number	n, o, t	3	0
ring-type	c, e, f, l, n, p, s, z	8	0
spore-print-color	k, n, b, h, r, o, u, w, y	9	0
population	a, c, n, s, v, y	6	0
habitat	g, l, m, p, u, w, d	7	0
class	e, p	2	0

2.1. tabula. Tabula parāda sēņu datu kopas visas iezīmes un klasi (class), to iespējamās vērtības, unikālo vērtību skaitu un tukšo ierakstu skaitu.



2.1. att. Sēņu datu kopas iezīmju kategoriskās vērtības

Visas raksturojošās iezīmes satur burtu vērtības, kuras reprezentē kategoriskas vērtības. Šādus datus nav iespējams vērtīgi apstrādāt ar izmantoto bibliotēku palīdzību, tāpēc tās tiek pārvērstas par skaitliskām vērtībām - katram skaitlim atspoguļojot citu kategoriju. Ar skaitliskām vērtībām pietiek, lai izmantotu VI algoritmu, taču lai apmācītu un novērtētu BBB un MC-D algoritmus ir nepieciešams veikt kategorisko vērtību iegulšanu. Kategorisko vērtību iegulšana ir populāra metode kategorisku datu apstrādei. Neironu tīkli nevar tieši apstrādāt kategoriskus datus, tādēļ nepieciešams šos datus pārveidot skaitliskā formātā. Tajā katrai unikālai kategorijai tiek piešķirta bināra vektora reprezentācija, kur visi elementi ir nulle, izņemot vienu, kas atbilst šai specifiskai kategorijai un ir atzīmēts ar vieninieku. Ja iezīme satur n unikālas vērtības, tā tiks pārvērsta par n vērtību tenzoru, lai ietvertu katru iespējamo kategoriju, kā apmācāmu vērtību. Tās attēlotu darbības principu var apskatīt tabulā 2.2.

Iezīme	Apstrādājamā vērtība
A	[1,0,0]
B	[0,1,0]
C	[0,0,1]

2.2. tabula. Tabula demonstrē iezīmju iegulšanas principu iezīmei ar 3 kategorijām

Datu kopa tiek sadalīta apmācības un testēšanas datu kopās. Tas tiek darīts, nejauši atlasot noteikto daudzumu ierakstu identifikatoru un atlasot datus divās grupās. Datu kopas izmantošanai binārās klasifikācijas problēmai, nepieciešams nodrošināt, ka dati apmācības un testēšanas kopā ir vienādā attiecībā. Tiek izveidota funkcija, lai atlasot apmācības un testa datus tiktu nodrošināts, ka abās datu kopās būtu līdzīgs skaits indīgu un ēdamu sēņu.

2.2. Ievaddatu jaukšana

Lai veiktu eksperimentus dažādos datu trokšņu apstākļos, tiek izmantota ievaddatu jaukšana. Ievaddatu jaukšana notiek, izmainot kategorisko datu vērtības apmācības datu kopai. To iespējams darīt pārveidotajām skaitliskajām vērtībām, kas iepriekš aprakstītas. Ievaddatu jaukšanu var veikt gan ievaddatu parametriem, gan ievaddatu klasēm, kas šajā gadījumā ir sēņu klasifikācija - ēdama vai neēdama. Eksperimenti tiek veikti gan jaucot iezīmes, gan klases, kā arī jaucot abus kopā. Šo eksperimentu vajadzībām tiek izveidota programma, kas balstoties uz norādīto % sajauc nepieciešamos apmācības datu vērtības. Process, kā tas tiek darīts ir sekojošs: klasēm, saturot tikai 0 vai 1, noteiktajam % ierakstu nejauši tiek nomainīta uz pretējo vērtību. Noteiktajam % ierakstu, katrai no iezīmēm, tiek nejauši izvēlēti indeksi . Tiem tiek nodzēstas orģinālās vērtības un aizvietotas ar kādu citu no konkrētajai iezīmei pieejamajām vērtībām.

Tiek veikti 6 eksperimenti:

1. Ievaddatu jaukšana netiek izmanīti.
2. Ievaddatu jaukšana tiek veikta 10% ierakstu.
3. Ievaddatu jaukšana tiek veikta 20% ierakstu
4. Ievaddatu jaukšana tiek veikta 30% ierakstu.
5. Ievaddatu jaukšana tiek veikta 40% ierakstu.
6. Ievaddatu jaukšana tiek veikta 50% ierakstu.

Tātad apvienojot kopā 6 dažādos datu jaukšanas apjomus, 3 dažādos ievaddatu jaukšanas veidus un 3 dažādos algoritmus, kopā sanāk 54 (6x3x3) atsevišķi algoritmu izpildījumi.

2.3. Salīdzināšanas protokols

Visiem trim algoritmiem ir izmaināmi hiperparametri, kas ietekmē algoritma veiktspēju, tomēr tie atšķiras ar modeļu arhitektūru. Vienīgie kopīgie hiperparametri visiem trim algoritmiem ir procents izmainīto marķējumu (ieejas vai izejas datiem), apmācības / testēšanas datu sadalījums un izlases izmērs jeb paraugu skaits, kas tiek ģenerēti, lai izveidotu vērtību sadalījuma, no kura izriet gala rezultāts.

Algoritmu darbības efektivitāte un precizitāte ir būtiski ietekmēta ne tikai no pašu algoritmu būtības un ievades datu kvalitātes, bet arī no noteiktiem kontrolējamajiem mainīgajiem. Visiem šeit apskatītajiem algoritmiem ir divi kopīgi kontrolējamie mainīgie: procentuāla izmaiņa ievades datu klasēs un iezīmēs. Šo mainīgo vērtības var ievērojami ietekmēt algoritma darbības rezultātus, tādēļ to optimālā vērtība ir jānosaka katram konkrētajai datu kopai un uzdevumam individuāli.

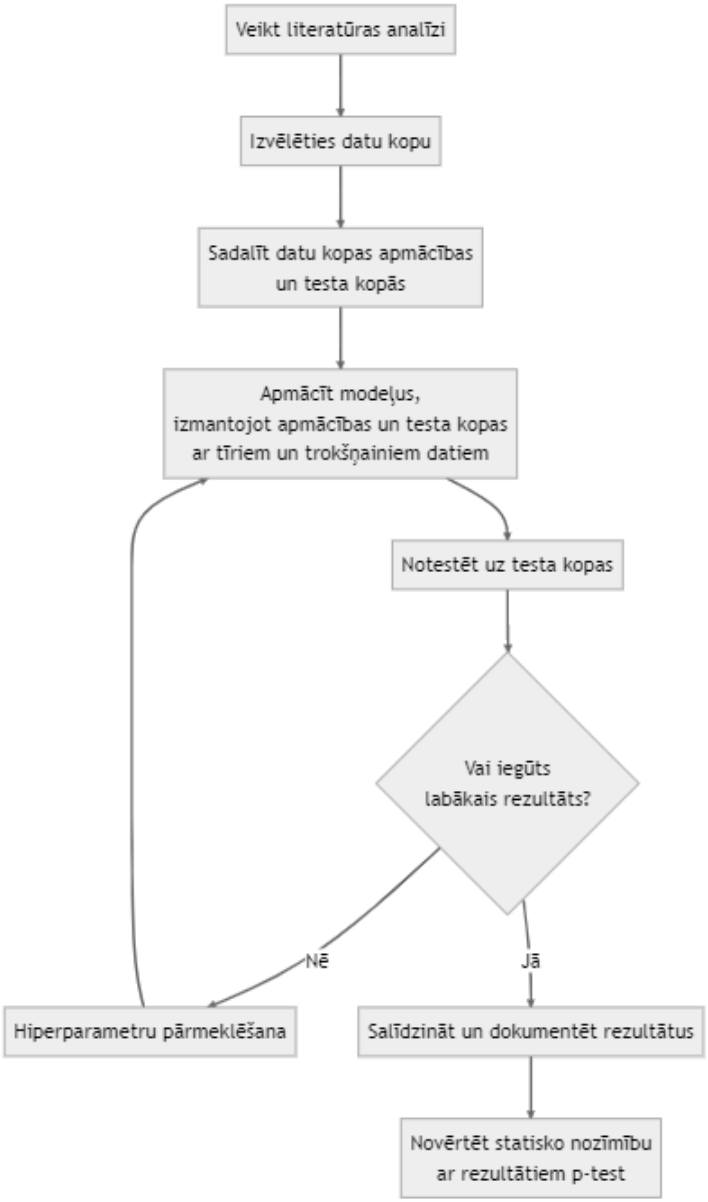
Visi algoritmi apvieno arī kopīgus hiperparametrus. Tie iekļauj apmācības un testēšanas datu sadalījumu un paraugu skaitu. Apmācības un testēšanas datu sadalījums ietekmē algoritma spēju vispārināt mācību rezultātus uz nezināmiem datiem, kamēr paraugu skaits var ietekmēt apmācības efektivitāti un precizitāti.

BBB un MC-D algoritmiem ir papildus kopīgi hiperparametri. Tie ietver partiju izmēru, apmācības ātrumu, apmācības ciklu skaitu, kā arī pirmā un otrā slāņa neironu skaitu. Šie parametri ievērojami ietekmē algoritmu apmācības procesu un galējo modeļa precizitāti. Lai optimizētu šo algoritmu darbību, ir nepieciešams rūpīgi regulēt šos hiperparametrus.

Tāpat arī VI algoritms ir papildināts ar specifiskiem hiperparametriem, kas ietver optimizācijas iterāciju skaitu, nejaušo izlašu skaitu un sākotnējo varbūtības sadalījumu ar μ un σ vērtībām. Šo hiperparametru optimizēšana ir būtiska VI algoritma efektivitātes un precizitātes nodrošināšanai.

Visbeidzot, BBB un MC-D algoritmiem ir vēl specifiski hiperparametri. BBB algoritmam tie ir šķērsentropijas kļūdas svars un KL novirze kļūdas svars, savukārt MC-D algoritmam ir caurkrites procents. Arī šo hiperparametru optimizēšana ir būtiska un tiek veikta, pirms algoritmi tiek salīdzināti.

Algoritmu salīdzināšana notiek strukturēti un sistemātiski. Pēc literatūras analīzes un datu kopas izvēles, algoritmi tiek implementēti Python programēšanas valodā. Pirmais solis visu algoritmu izveidē ir datu kopas sadalīšanas apmācības un testēšanas kopās. Ņemot vērā algoritmu lielo hiperparametru skaitu, ir svarīgi veikt hiperparametru pārmeklēšanu ar mērķi atrast kombināciju šiem hiperparametriem, kas izveido rezultātu ar vislielāko precizitāti datu klasificēšanā. Tātad, izmantojot implementētos algoritmus, katrs tiek atkārtoti izpildīts ar dažādām kombinācijām hiperparametru (bez trokšņa pievienošanas šajā brīdī, jo mērķis nav novērtēt stabilitāti, bet iegūt salīdzināmus algoritmus). Kad lielākās precizitātes hiperparametru kombinācijas ir iegūtas, eksperimenti tiek veikti un rezultāti tiek ievākti, salīdzināti un aprakstīti šī darba sadaļā "Rezultāti". Šī strukturētā salīdzināšanas metodi ir vizuāli parādīta attēlā 2.2.



2.2. att. Salīdzināšanas metodoloģijas shēma

Kā iepriekš aprakstīts, algoritmiem BBB un MCD ir vairāki kopējie hiperparametri. Tas ir tādēļ, ka abi ir implementēti, izmantojot mākslīgo neironu tīklu arhitektūru. Tas nozīmē, ka neironu tīkla slāņu skaits, apmācības ātrums un apmācības ciklu skaits, kā arī izmantotās aktivizācijas funkcijas ir maināmi hiperparametri. Šie hiperparametri tiek apskatīti tuvāk. Neironu tīkla slāņu skaits un katrā slānī esošo neironu skaits ir maināms, pievienojot vai noņemot tīkla slāņus un to neironu skaitus izmainot, lai izejas skaits vienā slāni sakristu ar iejas neironu skaitu nākošajā. Apmācības ātrums tiek mainīts no 1 līdz 0.0001. Apmācības ciklu skaits tiek reducēts, līdz grafika vizuāli saskatāmas konverģences brīdim. Aktivizācijas funkcijas, kuras iespējams izvēlēties, ir vairākas.

Aktivizācijas funkcijas ir fundamentāli elementi neironu tīklos, tostarp dziļajos

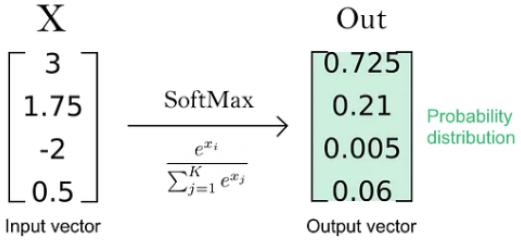
mācīšanās modeļos. Šīs funkcijas nosaka, cik spēcīgi konkrēts neirons reaģēs uz ienākošo signālu. Tās paņem ienākošo ieejas datu kopu un pārvērš to par izvadi, kas tiek nodota nākošajam slānim. Klasifikācijas uzdevumos, dažādas aktivizācijas funkcijas tiek izmantotas dažādos tīkla līmeņos un nodrošina dažādas priekšrocības. Slēpto līmeņu neironi parasti izmanto ReLU (Rectified Linear Unit) vai tās variantus, jo tie ir vienkārši un efektīvi. ReLU funkcija atgriež ieejas vērtību, ja tā ir pozitīva, un atgriež 0, ja tā ir negatīva. Formāli to var raksturot kā: $ReLU(x) = max(0, x)$. Izvades slānī tiek izmantotas tādas aktivizācijas funkcijas kā sigmoida vai softmax, kas var nodrošināt vērtības, kuras var interpretēt kā varbūtības.

Sigmoida funkcija atgriež vērtības intervālā (0,1), tādēļ tās izvade var tikt interpretēta kā varbūtība, kas ir noderīga binārās klasifikācijas problēmās. Sigmoida funkcija aprēķina ar formulu (2.1).

$$sigmoid(x) = 1/(1 + e^{-x}) \tag{2.1}$$

kur, $sigmoid(x)$ - sigmoid funkcijas vērtība;
e - Eilera skaitlis.

Softmax funkcija, kā apraksta autors, Gabriel Furnieles, (Furnieles, 2022) ir paplašinājums sigmoid funkcijai, jo tā sevī ietver to pašu algoritmu, tikai veiktu vairākiem ievades skaitļiem. Softmax funkciju izmanto vairāku klašu klasifikācijas problēmām, jo tā izveido vērtības kas summā ir 1, tātad var tikt uzskatītas par varbūtību sadalījumiem. Vizuāls piemērs apskatāms attēlā 2.3.



2.3. att. Attēls parāda sigmoid funkcijas darbību grupai vērtību, izcelts fakts, ka gala vērtības neizveido summā vēŗtību 1, tātad netiek uzskatītas par varbūtību sadalījumu.(Furnieles, 2022)

Dažādi statistisko metodes pārbauda, vai grupā vērtību ir vienāda varbūtību sadalījumu dispersija. Lēvena tests var tikt izmantots, lai pārbaudītu vai vairākas grupas ir ar vienādu dispersiju populācijā. Šajā darbā, tiek izmantots Lēvena tests, lai noskaidrotu, vai trīs algoritmu - VI, MC-D, BBB - rezultātu dispersijas ir statistiski nozīmīgi atšķirīgas mainīgā ieejas datu trokšņa līmeņa apstākļos. Nulles hipotēze šajā gadījumā būtu: ”Ieejas datu trokšņa līmeņa palielināšanās neietekmē algoritmu dispersijas starpību”. Alternatīvā hipotēze, savukārt, norāda uz pretējo: ”Ieejas datu trokšņa līmeņa

palielināšanās ietekmē algoritmu dispersijas atšķirību".

Lēvena tests ir balstīts uz datu grupu vidējo kvadrātisko novirzi, kas ir efektīvs līdzeklis, lai pārbaudītu vienādu dispersiju hipotēzi. P-vērtība tiek aprēķināta un salīdzināta ar iepriekš noteikto nozīmīguma līmeni (0.01). Ja p-vērtība ir mazāka par šo līmeni, mēs noraidām nulles hipotēzi, uzskatot, ka ieejas datu trokšņa līmeņa palielināšanās ietekmē algoritmu dispersiju atšķirību. Pretējā gadījumā nevar noraidīt nulles hipotēzi. Šis pētījums, pielietojot Lēvena testu, ļauj veikt pamatotus secinājumus par to, kā ieejas datu trokšņa līmeņa maiņa ietekmē algoritmu rezultātu dispersijas atšķirību, kas var kalpot kā vērtīga informācija modelēšanas procesa optimizācijā un rezultātu interpretācijā.

3. IMPLEMENTĀCIJA

Pirmkods tiek implementēts vienā vidē visiem algoritmiem, taču eksistē atšķirības to implementācijās. MCMC, MC-D un BBB algoritmiem tiek pielietota klasiskāka neironu tīkla arhitektūra, bet VI algoritmam tiek pielietota PyMC3 bibliotētkā pieejamā modeļa implementācija - dati tiek importēti, sadalīti testa un apmācības datu kopās.

Datu kopa tiek sadalīta apmācības un testa datu kopās, un tiek veiktas datu apstrādes operācijas, kas aprakstītas iepriekš. Programmatūras modeļi tiek apmācīti un pēc modeļa apmācības tiek veikta klasifikācija katram ierakstam no testa datu kopas. Rezultāti tiek ievākti. Šis process katram algoritmam ir aprakstīts tālāk.

Rezultātu apstrāde notiek visiem algoritmiem vienādi. Rezultātu dispersija un standartnovirze tiek aprēķinātas no modeļa rezultātiem un vidējā vērtība tiek uzskatīta par gala rezultātu, no kura tiek aprēķināti precizitātes mērījumi. Salīdzinot ar patiesajiem rezultātiem, testa datu kopā - tiek aprēķināti patiesie pozitīvie (tp), nepatiesie pozitīvie (fp), nepatiesie negatīvie (fn) un patiesie negatīvie (tn) rezultāti, izmantojot loģiskās operācijas. Tālāk tiek aprēķināta precizitāte, atsauce un F1 vērtība, izmantojot iepriekš aprēķinātos rezultātus. Pēc rezultātu ieguves tiek izmantota ”matplotlib” pirmkoda bibliotēka (Hunter, 2007) no kuras tiek veidoti grafiki šim darbam. Rezultātu kopējie grafiki apskatāmi pielikumos 4., 5., 6.

Neironu tīklus ir iespējams modelēt daudz dažādās programmēšanas valodās, taču visizplatītākā valoda ir Python. Tā satur dažādas koda bibliotēkas, kas ļauj izveidot mašīnapmācības algoritmus un tieši Beiesa neironu tīklus. Šajā darba izmantoju sekojošās bibliotēkas, lai modelētu salīdzināmos algoritmus.

1. PyTorch ir mašīnmācīšanās koda bibliotēka (Paszke, Gross et al., 2019)

2. NumPy ir bibliotēka priekš matemātisku darbību izpildes (Harris, Millman et al., 2020)

3. PyMC3 ir varbūtiskās programmēšanas bibliotēka priekš python programmēšanas valodas, kas ļauj veidot Beiesa modeļus ar daudz dažādām metodēm, to skaitā MCMC un VI. (Salvatier, Wiecki et al., 2015)

4. TorchBNN ir koda bibliotēka, priekš Beiesa modeļu izveides (S. Lee, Kim et al., 2022)

Eksperimentos veiktā hiperparametru pārmeklēšana liecina, ka visu 3 algoritmu efektivitāte ievērojami palielinās, palielinot izmantoto paraugu skaitu. Lielāks paraugu skaits nodrošina plašāku un detalizētāku informāciju par datu kopu, ļaujot algoritmiem veidot precīzākas prognozes. Tomēr jāņem vērā, ka paraugu skaita palielināšana saistās ar algoritmu izpildes laika pieaugumu. Katrs papildu apstrādātais paraugs prasa vairāk

laika un resursus, līdz ar to algoritma izpildes laiks būtiski palielinās ar pieaugošo paraugu skaitu. Šī pētījuma ietvaros izpildes laiks netika optimizēts, jo galvenais fokuss bija vērsts uz dispersijas un precizitātes salīdzināšanu, nevis uz algoritmu ātrdarbību.

3.1. Variāciju Inference

Pēc datu sagatavošanas tiek definēts Beiesa modelis ar funkciju pm.Model(), no bibliotēkas PyMC3. Šī funkcija satur vairākslāņu perceptrona modeļa definīciju, kurā ietilpst visi sākotnējie varbūtības sadalījumi un iespējamības funkcijas, kas ir nepieciešamas statistikas modelim. Šajā modelī normālsadalījumu funkcijas alfa un beta ir sākotnējā varbūtības sadalījuma vērtības, kas seko normālajam sadalījumam. Lineārās kombinācijas funkcija, kas ņem vērā šīs sākotnējā varbūtības sadalījuma vērtības un pazīmes no datu kopas, tiek izmantota, lai noteiktu μ vērtību. Tālāk iespējamības (likelihood) funkcija tiek definēta, izmantojot Bernoulli sadalījumu. Tad tiek veikta variāciju inferences tuvināšana, izmantojot funkciju pm.fit(), no bibliotēkas PyMC3. Šis solis ir kodols VI algoritma darbībā. Tas meklē optimālos parametrus, kas maksimizē ELBO un minimizē KL (Kullback-Leibler) novirze:

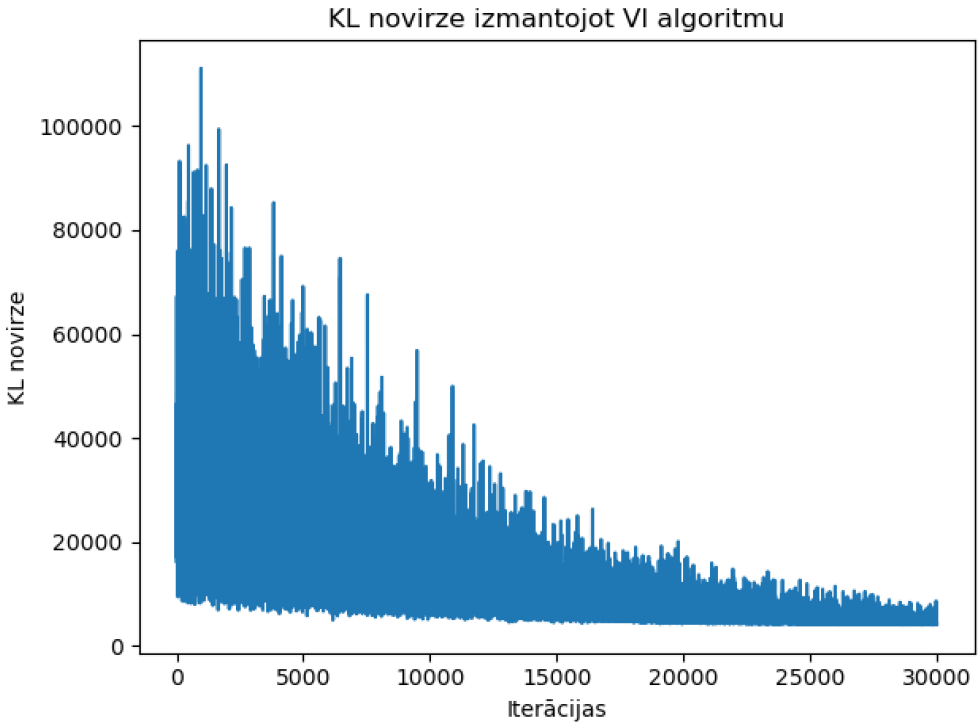
```
with pm.Model() as model:
    # Mūsu ievades dati
    X = pm.Data('X', train_data.iloc[:, 1:].T)

    # sākotnējā varbūtības sadalījuma mainīgie
    alpha = pm.Normal('alpha', mu=0, sd=0.1)
    beta = pm.Normal('beta', mu=0, sd=0.1, shape=(22,))

    # Lineārā kombinācija no iezīmēm
    mu = alpha + pm.math.dot(beta, X)

    # Ticamības funkcija
    pm.Bernoulli('y_obs', logit_p=mu, observed=train_data['class'])

    # Veicam variacionālo inferenci
    approx = pm.fit(method='advi', n=MODEL_ITERATION)

# Iegūstam novērtēto sadalījumu
trace = approx.sample(draws=DRAWS)

# Izvadam novērtētās vidējās vērtības
posterior_mean = approx.mean.eval()

with model:
    # Uzstādam testa datus
    pm.set_data({'X': test_data.iloc[:, 1:].T})

    # Veicam izlases veidošana no novērtēta sadalījuma
    ppc = pm.sample_posterior_predictive(trace, model=model, samples=POSTERIOR_SAMPLES)
```

```
    y = ppc['y_obs'].mean(axis=0)
    y_pred_vars = ppc['y_obs'].var(axis=0)

# Pārveidojam y par torch tensoru
tensor = torch.from_numpy(y)

# Piešķiram 0 un 1, balstoties uz robežvērtību 0.5
predicted_values = torch.where(tensor > 0.5, torch.tensor(1), torch.tensor(0))

# Patiesās klases vērtības
true_values = torch.tensor(test_data['class'].values)

# Aprēķinam rezultātus
precision, recall, f1, accuracy, tp, fp, fn, tn =
Helper.calculateAccuracy(predicted_values, true_values)
mean_variance = y_pred_vars.mean()
standard_deviation = math.sqrt(mean_variance)
}
```

Pēc tuvināšanas tiek veikta paraugu izvēle no gala vērtību sadalījuma. Šos paraugus izmanto prognozēšanai uz testa datu kopas. Prognozētās vērtības tiek salīdzinātas ar patiesajām vērtībām un no tām aprēķina precizitāti, pozitīvo p-vērtību pārklājumu (recall), F1 vērtību, dispersiju un stadartnovirzi, kā arī TP, FP, FN, TN rezultātus.

Izmantojot VI algoritmu, izvēlēto paraugu skaits ir būtisks faktors, kas ietekmē algoritma precizitāti un dispersiju. Lielāks paraugu skaits parasti noved pie precīzākas modeļa konverģences uz patieso pēcnovērojumu sadalījumu, jo vairāk informācijas tiek sniegts algoritmam. No otras puses, lielāks paraugu skaits var samazināt modelī ietverto novērojumu dispersiju. Kad tiek izmantots vairāk paraugu, mūsu modeļa novērtējumi kļūst stabilāki un mazāk atkarīgi no konkrētā parauga. Tādējādi, pieaugot paraugu skaitam, modelis kļūst mazāk jutīgs pret novērojumu novirzēm, kas noved pie zemākas dispersijas. Taču palielinot paraugu skaitu, tiek arī palielināts algoritma izpildes laiks.

Katrā algoritma izpildes reizē iterāciju parametri tiek izmainīti un Attēlā 3.1. tiek grafiski attēlota KL novirzes vērtība pret iterāciju skaitu - skaidri novērojams KL novirzes minimizēšanās.



3.1. att. KL novirzes izmaiņa VI algoritma izpildes laikā

Algoritma izpildei, pēc hiperparametru pārmeklēšanas, VI algoritmam tiek noteikts 30 000 optimizācijas iterāciju, nejaušo izlašu skaits - 200 un tiek izmantoti 200 paraugi, lai veiktu dispersijas aprēķinus. Sākotnējā varbūtības sadalījuma vērtības μ un σ ir vienādas ar 0 un 0.1 respektīvi. Apmācības datu apjoms ir 80% un testēšanas datu apjoms 20%.

3.2. Monte Karlo caurkrite

MC-D algoritmam tiek definēti modeļa slāņi un caurkrites slāņi izmantojot Pytorch bibliotēkā iebūvēto "torch.nn.Module" pirmkoda klasi. Slāņi algoritmā definēti ar Pytorch "torch.nn.Linear" klasi:

```
class mcd(nn.Module):
    def __init__(self, input_dim):
        super(mcd, self).__init__()
        self.fc1 = nn.Linear(input_dim, HIDDEN_LAYER_NODES_1)
        self.fc2 = nn.Linear(HIDDEN_LAYER_NODES_1, HIDDEN_LAYER_NODES_2)
        self.fc3 = nn.Linear(HIDDEN_LAYER_NODES_2, OUTPUT_DIMENSION)
        self.dropout = nn.Dropout(p=DROPOUT_RATE)

    def forward(self, x):
        x = torch.cat(x, dim=1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        x = F.relu(x)
```

```
x = self.dropout(x)
x = self.fc3(x)
x = F.softmax(x)
return x
```

Aktivizācijas funkcija, kas tiek pielietota ir ReLU. Algoritma izpildes laikā, dati tiek padoti algoritmam un izlaisti caur perceptrona slāņiem galā atgriežot tenzoru, kura izmēri ir (partijas izmērs x 2), kur 2 ir gala vērtība katram atsevišķajam ierakstam - vērtība, kas tiecas uz 1, un otra, kas tiecas uz 0, atbilstoši ēdamai vai indīgai klasifikācijai.

Algoritma izpilde notiek 2 etapos - apmācībā un testēšanā. Abi etapi sākas vienādi - dati tiek izvadīti caur tīkla slāņiem un katram ierakstam iegūts tenzors ar divām vērtībām, kas raksturo katru no gala klasēm. Abos etapos nākamais solis ir novērtēt rezultātu kļūdas vērtību, kas notiek, izmantojot šķērsentropijas kļūdas aprēķināšanu. Apmācības algoritmā nākamais solis ir atpakaļizplatīšanās, kuras laikā, modeļa neironu svari tiek izmainīti. Testēšanas etapā, katrai grupai ievaddatu, algoritms tiek izpildīts 100 reizes, lai iegūtu pārliecības mēru, dispersiju, iegūtajam rezultātam. Papildu kļūdas aprēķinam, testēšanas ciklā tiek noteikts modeļa pareizi kategorizēto ierakstu skaits un pievienots sarakstam, lai tiktu grafiski atspoguļots. Kad precizētais ciklu skaits ir sasniegts, pēdējā testēšanas ciklā rezultāti tiek apvienoti un ievākti, lai izveidotu kļūdu matricu.

Algoritma izpildei, pēc hiperparametru pārmeklēšanas, neironu tīklam tiek noteikti 3 lineārie slāņi - ieejas, apslēptais un izejas. ReLU funkcija tiek piemērota pirmajam un otrajam neironu slānim. Softmax aktivizācijas funkcija, tiek piemērota katrai vērtības izvades slānī. Caurkrites slāņi ar 70% caurkriti tiek piemēroti pēc ReLU funkcijas slāņiem. Neironu skaits pirmajam slānim ir 200, otrajam slānim ir 300 un trešajam ir 2, atbilstot datu kopas klašu skaitam - indīga vai ēdama. Apmācības ātrums ir 0.001 un apmācīto ciklu skaits ir 30. Apmācības datu apjoms ir 80% un testēšanas datu apjoms 20%. Partiju lielums ir 128 un tiek izmantoti 50 paraugi, lai veiktu dispersijas aprēķinus.

3.3. Beies caur atpakaļizplatīšanos

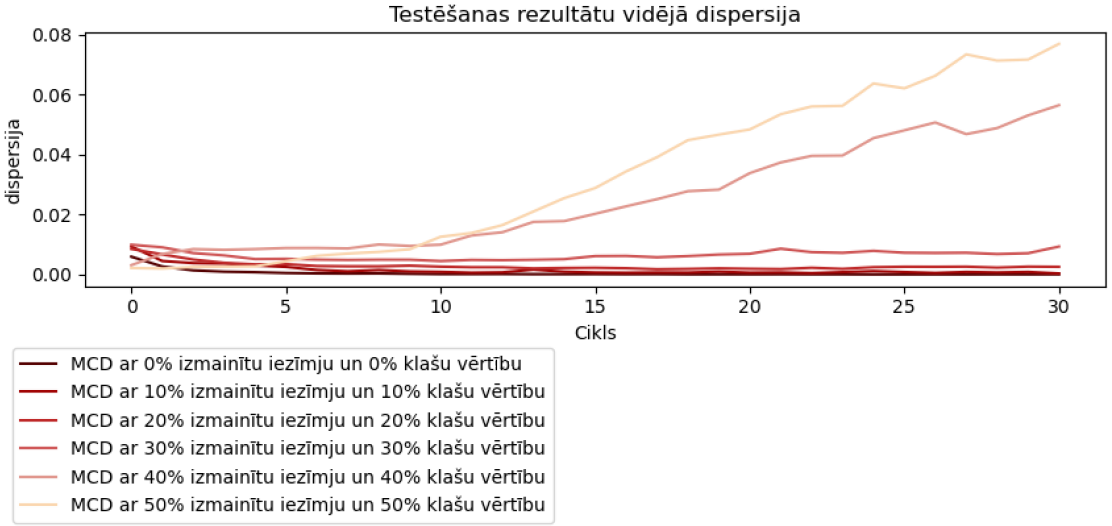
Līdzīgi kā MC-D algoritmā, modelis definēts izmantojot Pytorch ”Module” klasi, taču izmantotais lineārais transformācijas slānis ir izmantots no ”torchbnn” pirmkoda bibliotēkā pieejamās ”BayesLinear” funkcijas un aktivizācijas funkcija ir ReLU. Atšķirībā no MC-D algoritma, katram slānim tiek definētas netikai ievades datu izmēra un izvades datu izmēra mainīgie, bet arī μ un σ vērtības, kas nosaka ar kādu sadalījumu definēt modeļa svarus. Veiktie eksperimenti ir parādījuši, ka, izmantojot vairāk kā 4 slāņus tīklā, rezultāti katrai partijai sāk kļūt līdzīgi, vienalga kādi ievaddati ir izmantoti. Tas skaidrojams ar to, ka vienādas vērtību sadalījums izmantots jebkādiem ievaddatiem atkārtoti konverģēs uz vienādām vērtībām. No eksperimentiem secināts, ka visefektīvākais slāņu skaits BBB algoritmam ir 2 slāņi:

```
class BNN(nn.Module):
    def __init__(self, input_dim):
        super(BNN, self).__init__()
        self.fc1 = bnn.BayesLinear(prior_mu=PRIOR_MU_FIRST_LAYER,
        prior_sigma=PRIOR_SIGMA_FIRST_LAYER, in_features=input_dim,
        out_features=HIDDEN_LAYER_NODES_1)

        self.fc2 = bnn.BayesLinear(prior_mu=PRIOR_MU_FIRST_LAYER,
        prior_sigma=PRIOR_SIGMA_FIRST_LAYER, in_features=HIDDEN_LAYER_NODES_1,
        out_features=OUTPUT_DIMENSION)

    def forward(self, x):
        x = torch.cat(x, dim=1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.softmax(x, dim=1)
        return x
```

Apmācības etaps BBB algoritmam atšķiras no MC-D algoritma ar to, ka papildu šķersentropijas kļūdai, tiek aprēķināta arī KL novirzes vērtība, un, saskaitot to ar šķērsentropijas kļūdu, algoritms tiek optimizēts, lai uzlabotu šo abu vērtību kopējo rezultātu. Testa etaps notiek tāpat kā MC-D algoritmam - tiek veikta kļūdu aprēķināšana, pareizi kategorizēto ierakstu saskaitīšana un kļūdas matricas izveide.

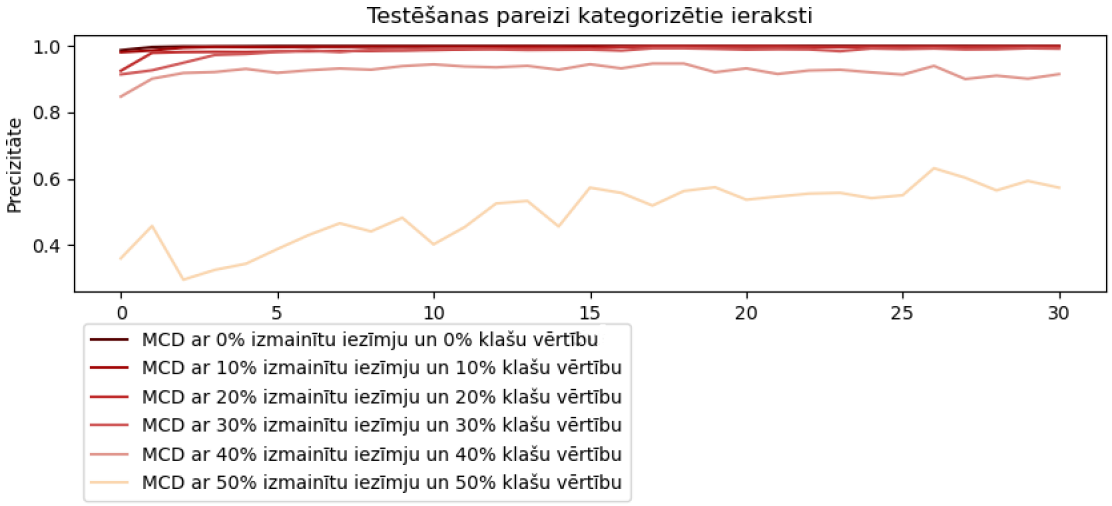
Algoritma izpildei, pēc hiperparametru pārmeklēšanas, neironu tīklam tiek noteikti 2 lineārie slāņi - ieejas un izejas. ReLU funkcija tiek piemērota tikai pirmajam neironu slānim. Softmax aktivizācijas funkcija, tiek piemērota katrai vērtības izvades slānī. Neironu skaits pirmajam un otrajam slānim ir 200 un 2 respektīvi, pēdējam slānim atbilstot datu kopas klašu skaitam - indīga vai ēdama. Vertības μ un σ visiem slāņiem ir vienādas - 0 un 0.1 respektīvi. KL novirzes kļūdas svars ir 0.3 un šķērsentropijas kļūdas svars ir 0.7. Apmācības ātrums ir 0.001 un apmācīto ciklu skaits ir 30. Apmācības datu apjoms ir 80% un testēšanas datu apjoms 20%. Partiju lielums ir 64 un tiek izmantoti 100 paraugi, lai veiktu dispersijas aprēķinus.

Abus neironu tipa algoritmus, MC-D un BBB, ir iespējams salīdzināt to rezultātus caur apmācības cikliem. Grafikos novērojams (3.2.att., 3.4.att.), ka abiem algoritmiem dispersiju absolūtās vērtības ir līdzīgas, taču to izmaiņa apmācības laikā ir atšķirīga. Šķietami MC-D algoritmam pirmajos pāris apmācības ciklos tās pieaug un konverģē, bet BBB algoritmam tās strauji samazinās un konverģē. Šādu efektu varēja novērot dažādos hiperparametru konfigurācijas līmeņos, BBB algoritmam tipiski dispersija apmācības laikā pazeminājās, bet MC-D algoritmam varēja noverot tās palielināšanos atsevišķos eksperimentos.

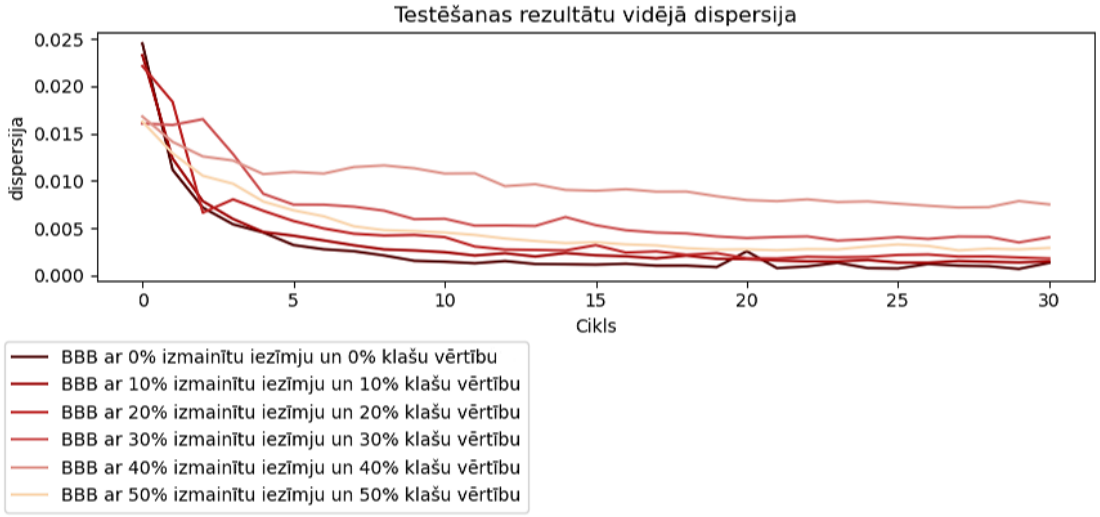


3.2. att. Attēls parāda dispersijas maiņu apmācības procesā MC-D algoritmam

MCD algoritmam novērojams, ka augstākos trokšņa apstākļos dispersijas maiņa ir straujāka, ciklu skaitam palielinoties. Apskatot šķietami ātri konverģējošos rezultātus precizitātes datiem grafikā 3.3. var spriest, ka zemāku ciklu skaits augsta trokšņa apstākļos varētu nereprezentēt patieso nenoteiktības situāciju rezultātos.



3.3. att. Attēls parāda precizitātes maiņu apmācības procesā MCD algoritmam



3.4. att. Attēls parāda dispersijas maiņu apmācības procesā BBB algoritmam

Aplūkojot dispersijas diagrammu attēlā 3.4., var novērot, ka izmantotie 30 cikli BBB algoritmam ir vairāk nekā pietiekami, lai sasniegtu dispersijas konverģenci. Salīdzinot šādus grafikus ar precizitātes grafikiem, lietotāji var optimizēt algoritmu darbības laiku, samazinot pielietoto ciklu skaitu. Algoritmu rezultātu maiņu apmācības procesā, var aplūkot pielikumos - 7., 8., 9., 10., 11., 12.

4. REZULTĀTI

Veikti eksperimenti, kuros tika salīdzināti Variāciju Inferences, Monte Karlo caurkrite un Beiess caur atpakaļizplatīšanos. Sākumā tiek salīdzinātas algoritmu precizitātes. Rezultāti no eksperimentiem liecina, ka neironu tīklu metodes algoritmi, BBB un MC-D, klasificē kategorizētos ierakstus ar vislielāko precizitāti. Tabulās 4.1., 4.2., 4.3. ir apkopota katra algoritma precizitāte katrā jaukto iezīmju un klašu daudzuma scenārijā ar vidējās standartnovirzes vērtību. Variāciju inferences algoritma precizitāte šajos novērojumos ir zemākā visos eksperimentos. Šie novērojumi liecina, ka neironu tīklu metodes, kas izmantotas MC-D un BBB algoritmos ir efektīvākas klasifikācijas uzdevumā.

Algoritms	**Jauktie marķējumi klasēm un iezīmēm**					
	0%	10%	20%	30%	40%	50%
MC-D	100 ± 0.00	99.94 ± 0.03	99.79 ± 0.05	98.18 ± 0.10	90.67 ± 0.20	59.49 ± 0.27
VI	90.76 ± 0.33	88.49 ± 0.41	87.62 ± 0.45	85.71 ± 0.48	76.60 ± 0.49	51.17 ± 0.50
BBB	99.70 ± 0.05	98.35 ± 0.06	98.05 ± 0.06	94.60 ± 0.07	92.93 ± 0.11	55.29 ± 0.11

4.1. tabula. Tabula parāda katra algoritma precizitāti datu klasificēšanā ar standartnovirzi, dažādiem % jauktu klašu un iezīmju marķējumu.

Algoritms	**Jauktie marķējumi klasēm**					
	0%	10%	20%	30%	40%	50%
MC-D	100 ± 0.01	100 ± 0.03	99.94 ± 0.04	98.71 ± 0.08	94.72 ± 0.10	50.45 ± 0.09
VI	90.76 ± 0.33	91.87 ± 0.39	89.78 ± 0.44	90.95 ± 0.48	83.93 ± 0.49	53.39 ± 0.50
BBB	99.82 ± 0.05	99.70 ± 0.05	99.70 ± 0.06	98.38 ± 0.09	96.91 ± 0.12	57.20 ± 0.09

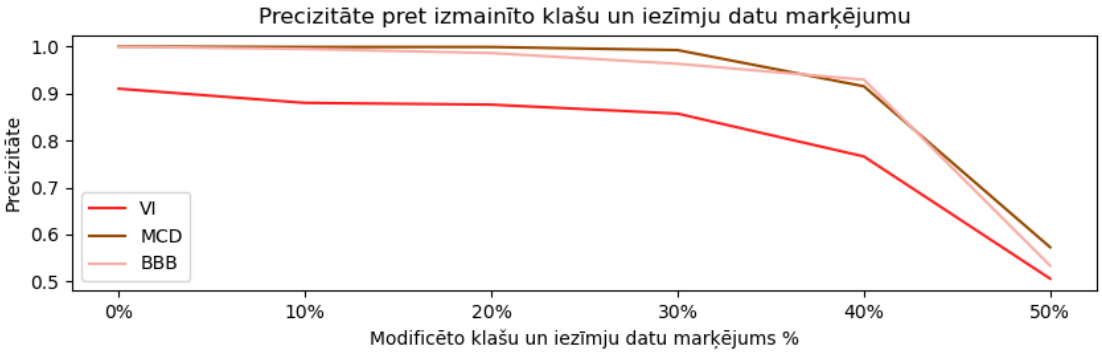
4.2. tabula. Tabula parāda katra algoritma precizitāti datu klasificēšanā ar standartnovirzi, dažādiem % jauktu klašu marķējumu.

Visaugstākā precizitāte, augsta trokšņa apstākļos tiek sasniegta scenārijā, kur troksnis ir pievienots tikai iezīmēm (4.3. tabulas 7.kolonna). Tomēr precizitātes rezultāti ir ļoti līdzīgi scenārijos, kur troksnis ir pievienots tikai klasēm vai gan iezīmēm, gan klasēm. Šis novērojums, liek domāt par atšķirību starp klašu un iezīmju jaukšanas ietekmi algoritma veiktspējai - troksnis izvaddatos iespējams ietekmē algoritma veiktspēju vairāk kā troksnis ievaddatos. Iespējams, ka pielietotai datu kopai saturot 23 dažādas iezīmes, algoritmi ir relatīvi noturīgāki pret trokšņa ieviešanu iezīmēm, jo algoritmi ir spējīgi izmantot informāciju kas ietverta starp nejauktajām iezīmēm. It īpaši šāds skaidrojums palīdz izskaidrot MC-D algoritma salīdzinoši augsto stabilitāti jaukto iezīmju scenārijā, kas apskatāms visās kolonnās tabulā 4.3., jo caurkrite, kas iebūvēta algoritmā, ļauj modelim nepievērst lielu nozīmi atsevišķām iezīmēm, bet izmantot informāciju, kas latentos mainīgos, kas paslēpti datu kopā.

Algoritms	Jauktie marķējumi iezīmēm					
	0%	10%	20%	30%	40%	50%
MC-D	100 ± 0.01	99.94 ± 0.02	99.88 ± 0.02	99.88 ± 0.03	99.34 ± 0.05	95.77 ± 0.07
VI	91.63 ± 0.33	88.49 ± 0.35	87.81 ± 0.36	86.88 ± 0.37	86.08 ± 0.37	84.36 ± 0.38
BBB	99.70 ± 0.05	99.04 ± 0.05	98.32 ± 0.05	95.68 ± 0.07	94.46 ± 0.06	92.33 ± 0.06

4.3. tabula. Tabula parāda katra algoritma precizitāti datu klasificēšanā ar standartnovirzi, dažādiem % jauktu iezīmju marķējumu.

Palielinoties datu kopas troksnim, algoritmu precizitāte samazinās visiem algoritmiem. Tas ir novērojams gan ar tīriem datiem, gan ar trokšņainiem datiem visos trīs dažādos trokšņa pievienošanas veidos. Attēlā 4.1. var grafiski apskatīt precizitātes uzvedību datu troksnim palielinoties visiem trim algoritmiem. Šādi rezultāti ir sagaidāmi, jo palielinoties troksnim apmācības datu kopā, algoritms var sākt uzsvērt nejaušas korelācijas, nevis patiesas atkarības datu kopā. Šie rezultāti ir ilustratīvi tam, cik svarīgi ir ņemt vērā datu troksni, algoritma veiktspējas uzlabošanā.



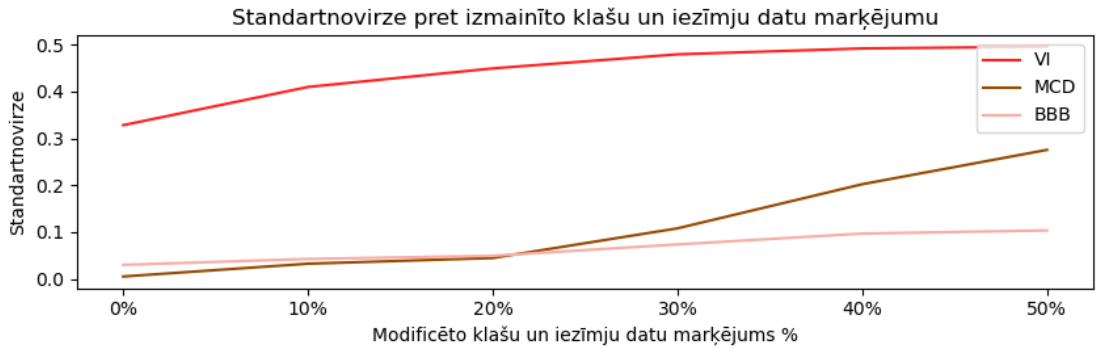
4.1. att. Attēls parāda precizitātes maiņu pret pievienotā trokšņa daudzumu visiem trim algoritmiem

Vēl precizitātes rezultātos (attēlā 4.1.) izceļas VI algoritma zemā precizitāte zemos datu trokšņa apstākļos. It īpaši jāizceļ gadījums, kad jauktas ir tikai iezīmes - VI algoritma precizitāte ir zemāka ar nejauktiem datiem, nekā BBB un MC-D ir 50% jaukto iezīmju scenārijā. VI algoritma zemo precizitāti var skaidrot ar to, ka iespējams VI implementācija, kas pieejama PyMC3 pirmkoda bibliotēkā, nav spējīga pilnīgi satvert datu kopas nelineārās saistības starp iezīmēm un iespējams šādam uzdevumam, ir vajadzīga cita VI algoritma implementācija.

Tiek salīdzināti 3 dažādo trokšņa ieviešanas scenāriji. Kā iepriekš aprakstīts, iezīmju jaukšanai ir šķietami mazāks iespaids uz algoritmu rezultātiem nekā klašu jaukšanai. Gadījumā, kad apmācības datu kopai tiek nejauši apmainītas binārās klašu vērtības, dati pazaudē ļoti daudz sakarības, kuras nepieciešams, lai veiktu jēgpilnus novērtējumus. Taču gadījumā, kad nejauši ir izmainītas iezīmju vērtības, atlikušo, neizmainīto iezīmju sakarības ar novērtējamo klasi, netiek pazaudētas. Balsoties uz šo, nav pārstei-

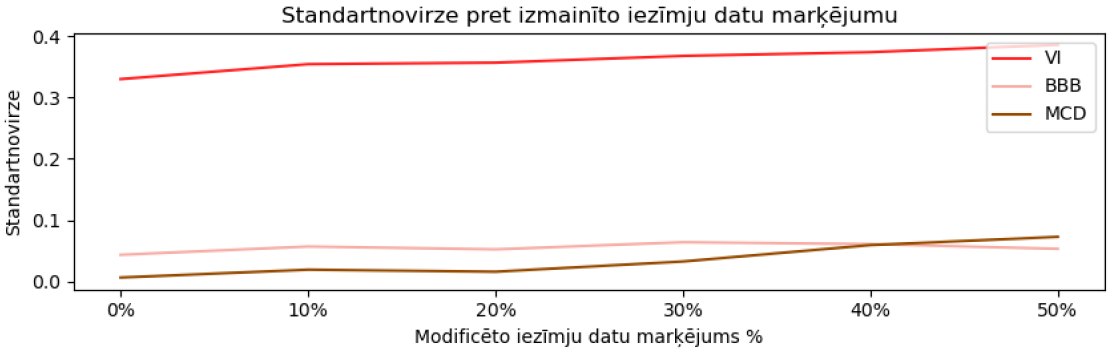
dzoši novērot (tabulas 4.1., 4.2.), ka algoritmu precizitātes nokritās līdz 50%, kad puse no klasēm ir izmainītas.

Apskatot rezultātos iegūtās standartnovirzes vērtības attēlā 4.2., ir novērojams, ka standartnovirzes vērtības ir atšķirīgas starp visiem algoritmiem. VI rezultātiem tā ir vislielākā, apmēram 10 reizes lielāka nekā BBB un novērojami lielāka par MC-D algoritmu. Šī standartnovirzes vērtība VI algoritmam atbilst ar to, ka tam ir bijusi viszemākā precizitāte - tātad biežāk kļūdoties algoritms var biežāk izvest vērtību tālāk no vidējās. Iespējams arī novērot, ka visiem algoritmiem, standartnovirze pieaug, pieaugot jaukto marķējumu daudzumam, kas liecina, ka algoritmi ir jūtīgi pret trokšņa daudzumu. VI un MC-D algoritmiem novērojams vislielākais standartnovirzes izmaiņas troksnim pieaugot. BBB algoritmam šķietami standartnovirze mainās mazāk un atsevišķos scenārijos, tabula 4.3. kolonna 5, tas palielinās pieaugot troksnim. Tas varētu liecināt par rezultātu stabilitāti, bet zinot, ka objektīvi troksnis ir pieaudzis un algoritms to neatspoguļo, samazinās standartnovirzes iegūšanas jēgu.



4.2. att. Attēls parāda standartnovirzes maiņu pret pievienotā trokšņa daudzumu visiem trim algoritmiem

Standartnoviržu vērtībās ir novērojamas atšķirības starp jaukto datu scenārijiem. Līdzīgi kā precizitātes datiem, gadījumā, kad jauktas ir tikai iezīmes (attēls 4.3.), standartnovirzes ir zemākas, kā tad, ja jauktas ir klases vai klases un iezīmes. Skaidrojums, iespējams, ir līdzīgs kā precizitātei - samazinoties datu ierakstu sakarībām ar klasēm, kurās tās kategorizētas, algoritmi paliek nedrošāki par katru klasificēto ierakstu.



4.3. att. Attēls parāda standartnovirzes maiņu pret pievienotā iezīmju trokšņa daudzumu visiem trim algoritmiem

Salīdzinot kļūdas matricu rezultātus, kas parādīti tabulās 4.4., 4.5., 4.6., iespējams novērot, ka eksistē atšķirības patiesi pozitīvo un patiesi negatīvo vērtību sadalījumā. Lai gan MC-D algoritms identificē visus datus pareizi, VI un BBB algoritmi nepareizi klasificē vairākas vērtības, par kurām var veikt interesantus novērojumus. Par pozitīvo vērtību tiek uzskatīta ēdama sēnes identifikācija un negatīvo vērtību tiek uzskatīta indīga sēne. BBB un VI algoritmiem ir novērojams, ka tie biežāk nepareizi klasificē ēdamas sēnes, nekā indīgas sēnes, kas, kā jau iepriekš aprakstīts, ir labvēlīgāk nekā alternatīva. Apskatot visus eksperimentu rezultātus TP, FP, FN, TN vērtībām (pielikumi 1., 2., 3.), var redzēt, ka visi algoritmi biežāk kļūdas klasificējot indīgās sēnes, izņemot MC-D algoritmu, kas jaukto iezīmju un klašu gadījumā, biežāk nepareizi klasificē ēdamās. Kāpēc algoritmu šis ir novērojams, nav skaidrs, jo kā pieminēts iepriekš, datu kopa ir relatīvi balansēta. Nepieciešams veikt papildu izmeklējumus, lai izskaidrotu šos rezultātus.

		Īstās vērtības	
		Pozitīvi	**Negatīvi**
Paredzētās vērtības	**Pozitīvi**	696	34
	Negatīvi	116	778

4.4. tabula. Kļūdu matrica priekš VI algoritma ar tīriem apmācības datiem

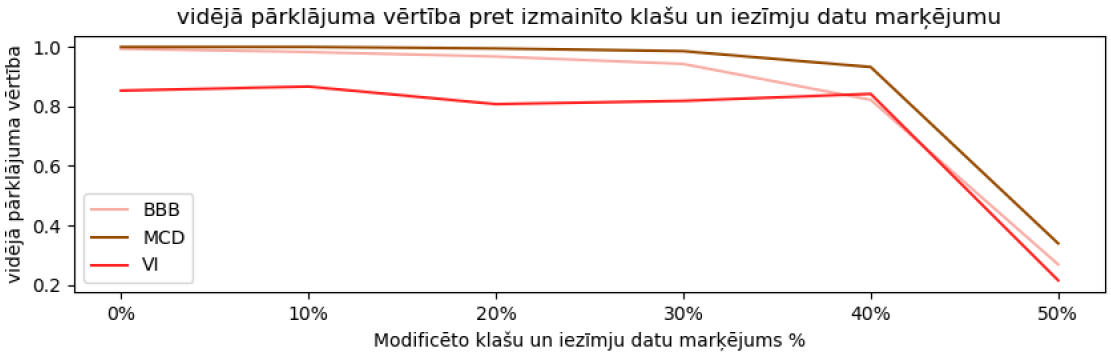
		Īstās vērtības	
		Pozitīvi	**Negatīvi**
Paredzētās vērtības	**Pozitīvi**	812	0
	Negatīvi	0	812

4.5. tabula. Kļūdu matrica priekš MC-D algoritma ar tīriem apmācības datiem

		Īstās vērtības	
		Pozitīvi	**Negatīvi**
Paredzētās vērtības	**Pozitīvi**	807	0
	Negatīvi	5	812

4.6. tabula. Kļūdu matrica priekš BBB algoritma ar tīriem apmācības datiem

Pārklājuma rezultāti, kas redzami attēlā 4.4., ir ļoti līdzīgi precizitātes grafikam, tomēr iespējams varam novērot saistītu rezultātu ar kļūdas matricām - pārklājums sevī ietver vairāk informāciju par pozitīvi novērtētajām vērtībām, ignorējot nepareizi identificēto negatīvo gadījumu skaitu. Var redzēt, ka BBB algoritmam pie 40% jaukto klašu un iezīmju ir līdzīgāka vērtība VI algoritmam, nekā MC-D algoritmam, kā tas novērojams precizitātes rezultātos.



4.4. att. Attēls parāda pārklājuma maiņu pret pievienotā trokšņa daudzumu visiem trim algoritmiem

4.1. Rezultātu pārbaude

Rezultāti tiek statistiski analizēti, lai novērtētu, cik statistiski nozīmīgi ir iegūtie rezultāti. Lēvena tests tiek izmantots, lai pārbaudītu hipotēzi, ka vairāku grupu dispersijas ir vienādas. Nulles hipotēze šajā gadījumā ir tāda, ka katra no algoritmu dispersijām pie dažādiem trokšņa līmeņiem ir vienādas. Tātad Lēvena testa mērķis šajā eksperimentā ir noteikt, vai algoritmu dispersijas atšķiras no trokšņa līmeņiem, uz kuriem tie tika apmācīti. Lēvena testu rezultāti, apskatāmi tabulā 4.8. Visiem trīs algoritmiem p-vērtības ir zem noteiktā nozīmīguma līmeņa (0.01), kas norāda, ka mēs varam noraidīt nulles hipotēzi, proti, algoritmu dispersijas atšķiras atkarībā no trokšņa līmeņa. Tātad mūsu datus ietekmē trokšņa līmenis. Šādi paši rezultāti tika novēroti arī tikai klašu vai tikai iezīmju jaukšanas scenārijos.

Algoritms	P-vērtība
MC-D	0.00
BBB	1.3E-28
VI	0.0

4.7. tabula. Tabula parāda P-vērtības, kas aprēķinātas no variances vērtībām, izmantojot Levēna testu.

Tālāk tika veikti Lēvena testi, lai salīdzinātu datus starp algoritmiem dažādos trokšņa līmeņos. Katrā trokšņa līmenī tika salīdzinātas dispersijas starp algoritmiem. Šo testu mērķis bija noskaidrot, vai algoritmu dispersijas atšķiras ne tikai atkarībā no trokšņa līmeņa, bet arī starp pašiem algoritmiem pie noteikta trokšņa līmeņa. Pēc Lēvena testa veikšanas, iegūtie rezultāti, apskatāmi tabulā 3.9., parāda, ka nulles hipotēzi ir jānoraida - algoritmu dispersijas pie noteikta trokšņa līmeņa nav vienādas, jo p-vērtības ir zem noteiktā nozīmīguma līmeņa (0.01). Šī analīze liecina, ka algoritmu dispersiju atšķirības ir statistiski nozīmīgas ne tikai atkarībā no trokšņa līmeņa, bet arī starp pašiem algoritmiem.

Trokšņa daudzums	Salīdzinātie algoritmi	Lēvena testa vērtība	p-vērtība
0%	VI & BBB	2931.657402	0
0%	VI & MC-D	3377.048314	0
0%	BBB & MC-D	130.076677	1.43E-29
10%	VI & BBB	3149.057741	0
10%	VI & MC-D	3622.432529	0
10%	BBB & MC-D	39.77601811	3.23E-10
20%	VI & BBB	1456.679658	1.27E-263
20%	VI & MC-D	1537.645287	1.15E-275
20%	BBB & MC-D	9.828975059	0.00173
30%	VI & BBB	473.5928215	3.96E-98
30%	VI & MC-D	65.74270766	7.21E-16
30%	BBB & MC-D	72.85921968	2.11E-17
40%	VI & BBB	491.1451873	1.87E-101
40%	VI & MC-D	1178.055974	1.53E-220
40%	BBB & MC-D	1965.906244	0
50%	VI & BBB	279.3652158	3.17E-60
50%	VI & MC-D	2181.649903	0
50%	BBB & MC-D	2807.818624	0

4.8. tabula. Tabula parāda P-vērtības un Levēna testu vērtības, salīdzinot algoritmu rezultātus noteiktos % jauktu klašu un iezīmju.

Vismazākā atšķirība ir novērojama BBB un MC-D algoritmiem mazu trokšņu apstākļos, kur abi algoritmi spēja veikt ļoti augstas precizitātes klasifikāciju. Ņemot vērā, ka abi algoritmu rezultāti veido augstu precizitāti un dispersiju un pie 20% trokšņa novērojama 0.00173 p-vērtība (tabula 4.8., kolonna 4, rinda 10), jādomā, ka algoritma izmantošanas izvēle varētu tikt veikta balstoties uz citiem kritērijiem, kā piemēram, izpildes laiks, implementācijas sarežģītība u.c.

5. TĀLĀKIE PĒTĪJUMI

No veiktajiem eksperimentiem iespējams apsvērt veidus, kā papildu būtu iespējams salīdzināt Beiesa algoritmu noturīgumu pret troksni jeb stabilitāti. Tālākie pētījumi varētu veikt pārbaudi, vai algoritmiem novērotā jūtība pret ievaddatu troksni ir novērojama arī citām datu kopām. Sēņu datu kopa, kas izmantota šajā pētījumā ir balansēta, ar nelielu skaitu tukšu datu punktu, un tā satur daudz iezīmju no kurām algoritms var iegūt informāciju. Kā šie algoritmi strādā ar citām datu kopām būtu vērtīgi izpētīt, lai veiktu vispārējākus secinājumus.

Šajā darbā tika demonstrēts, kā MCD, BB un VI algoritmu darbība mainās, atšķirīgos datu tīrības apstākļos. Kā iepriekš aprakstīs, eksistē citas Beiesa metodes, kuras var izmantot pārliecības mēra noteikšanai, kuras būtu vērtīgi pievienot šim salīdzinājumam. Citi pētījumi varētu izmeklēt melnās kastes variāciju inferences vai strukturēta vidējā lauka variāciju inferences metodes, lai analizētu un plašāk izskaidrotu VI algoritma salīdzinājumu ar MC-D un BBB algoritmiem.

Svarīgi norādīt, ka šajā pētījumā tika apskatīta tikai trokšņa ietekme uz algoritmu precizitāti un to standartnovirzi. Citas faktori, piemēram, klasifikācijas uzdevuma sarežģītība vai apmācības datu apjoms, varētu arī ietekmēt šo algoritmu stabilitāti. Turpmākajos pētījumos vajadzētu izpētīt šos aspektus, lai sniegtu visaptverošāku izpratni par Beiesa neironu tīklu salīdzinošo veiktspēju un stabilitāti. Turklāt, lai gan mūsu pētījumā BBB un MC-D algoritmi parādīja līdzīgu veiktspēju visos testa apstākļos, turpmākā analīze varētu identificēt noteiktus scenārijus vai datu kopas, kurās viens algoritms pārsniedz otru. Šādas atziņas varētu būtiski ietekmēt algoritmu izvēli konkrētos lietošanas gadījumos.

Eksperimentu veikšanas laikā, radās ideja, veikt salīdzinājumus paraugu skaita ietekmei uz rezultātu dispersiju. Teorētiski, zemā paraugu skaitā, varētu sagaidīt lielāku dispersiju, jo, palielinoties paraugu skaitam, rezultāti pietuvotos patiesajam vērtību sadalījumam. Taču, lai apstiprinātu šo hipotēzi un noskaidrotu kuriem algoritmiem paraugu skaits vairāk ietekmē rezultātus, būtu nepieciešams veikt papildu pētījumus. Šis varētu būt vērtīgi, ja algoritma izpildes laiks ir nozīmīgs, jo kā eksperimenti parādīja, paraugu skaitam ir liela ietekme uz algoritmu izpildes laiku.

Eksperimentu veikšana iedvesmoja vēl vienu ideju, trokšņa analīzei, kas, nebūtu jābūt saistītai ar Beiesa tīkliem. Ņemot vērā konstatētās atšķirības jaukto klašu un iezīmju situācijās, rodas jautājums, vai mašīnmācīšanās algoritma tolerance pret datu troksni atšķiras no trokšņa avota. Vai atsevišķi algoritmi ir noturīgāki pret ieejas datu troksni nekā izejas datu troksni? Šajā pētījumā tiek iegūti rezultāti, kuri liecina, ka klašu jaukšanai ir lielāka ietekme nekā iezīmju jaukšanai, bet pētījumi, kuri tuvāk pievērstos šīs ietekmes izpētei, varētu palīdzēt spriest par potenciālajām izmaksām datu attīrīšanai ievades un izvades datiem. Lai nodrošinātu efektīvu resursu izmantošanu un optimā-

lu algoritmu darbību, ir būtiski atrast līdzsvaru starp precizitāti, dispersiju un izpildes laiku. Tas var prasīt eksperimentālu pieeju, lai noteiktu optimālo paraugu skaitu, kas nodrošina optimālākos rezultātus specifiskajām vajadzībām.

Veicot papildu izmeklējums būtu iespējams spriest par iespēju noteikt datukopas trokšņa līmeni ar kāda algoritma palīdzību. Šis pētījums parāda, ka trokšņa līmenim ir saistība ar algoritmu dispersiju, papildu pētījumi ir vajadzīgi, lai noskaidrotu, vai iespējams paredzēt datu kopas trokšņa līmeni balstoties uz dispersijas datiem no kāda algoritma. Tas varētu būt vērtīgi gadījumos, kad datu tīrīšanas izmaksas var būt dārgas un trokšņa daudzuma apzināšanās varētu būt noderīga informācija resursu ietaupīšanas vajadzībām.

6. SECINĀJUMI

Šis pētījums sniedzis pierādījumus par Beiesa algoritmu un Monte Karlo caurkrites metožu jūtību pret datu troksni gan to precizitātes, gan dispersijas vērtībās. Iegūtie rezultāti norāda uz statistiski nozīmīgām atšķirībām starp minētajiem algoritmiem - konkrēti, Monte Karlo caurkrites un Variāciju Inferences algoritmi pierāda paaugstinātu jūtību pret iekļauto datu troksni apmācības datos, salīdzinājumā ar Beiesa atpakaļizplatīšanis algoritmu. Monte Karlo caurkrites un Variāciju Inferences algoritmi, balstoties uz veiktajiem eksperimentiem, spēj precīzāk atspoguļot nenoteiktību rezultātos.

Šajā pētījumā tika konstatēts, ka MC-D un BBB algoritmi ir piemērotāki binārās klasifikācijas uzdevumam, dažādos trokšņa līmeņos. Savukārt VI algoritms, kura realizācija tiek veikta izmantojot atvērtā pirmkoda bibliotēku PyMC3, var prasīt papildu iepriekšēju datu priekšapstrādes soļus, lai mazinātu trokšņa negatīvo ietekmi uz tā darbības efektivitāti. Tomēr, šie secinājumi par VI algoritma vispārējo efektivitāti tiek uzskatīti par apstrīdamiem, ņemot vērā dažādās realizācijas metodes, kas ir pieejamas VI algoritmam. Ir iespējams, ka PyMC3 bibliotēkā piedāvātā VI algoritma implementācija nebija optimāli piemērota šim eksperimentam.

BBB un MC-D algoritmu precizitātes ir ļoti augstas zema trokšņa apstākļos, tomēr tikai MC-D algoritms ir spējīgs sasniegt 100% precizitāti arī apzināti ieviesta trokšņa apstākļos. VI algoritms veica klasifikācijas uzdevumu ar zemāku precizitāti, un ar hiperparametru pārmeklēšanu nav iespējams sasniegt tik augstu veiktspēju, kā to bija iespējams sasniegt izmantojot MC-D un BBB algoritmus.

Pētījumā tika konstatētas statistiski nozīmīgas atšķirības dispersiju vērtībās. VI un MC-D algoritmu dispersijas rezultāti bija būtiski ietekmēti, kad pieauga apmācības datu troksnis. Lai gan izmaiņas BBB algoritma standartnovirzē ir novērotas un novērtētas par statistiski nozīmīgām, to mazā izmaiņa, it īpaši, tikai iezīmju trokšņa gadījumā, ievieš šaubas par algoritma spēju efektīvi modelēt pārliecības pakāpi.

Eksperimentu rezultāti liecina, ka apskatītie algoritmi ir spējīgi nodrošināt augstu precizitāti un stabilu dispersijas vērtības līdz 40% jauktu apmācības datu situācijās. Tomēr, sasniedzot 50% jauktu klašu, algoritmu stabilitāte pilnīgi pazūd, precizitātei nokrītoties līdz aptuveni 50%. Papildus, no šiem datiem var secināt, ka visi trīs algoritmi uzrāda stabilāku darbību atsevišķi iezīmju jaukšanas scenārijā, nekā tikai klašu vai kombinētajā klašu un iezīmju jaukšanas scenārijos.

Eksperimentu rezultāti parāda, ka trokšņa ieviešana apmācības datu klasēs rada lielāku ietekmi nekā trokšņa ieviešana iezīmju līmenī. Tas tiek skaidrots ar izmantotās datu kopas īpašībām - plašo iezīmju skaitu, kas, pat jauktā scenārijā ir spējīgas saturēt vērtīgu informāciju. Tomēr, kad sajauktas 50% no visām iezīmēm, algoritmi ir spējīgi saglabāt augstu precizitātes līmeni ar relatīvi zemu standartnovirzi, kas nav novērojams apmācības datu klašu jaukšanas gadījumā. Tātad, trokšņa klātbūtne apmācības datu

kopas izvaddatos ir nozīmīgāka par trokšņa ieviešanu ievaddatos.

Līdz šim brīdim ir veikta reģistrācija konferencē ”7th International Conference on Innovations and Creativity”, kas notiks no 1. līdz 3. jūnijam, 2023. gadā, Liepājas Universitātē, Liepājā, Latvijā, kurā tiks prezentēts šis pētījums. Pēc recenzijas procesa pastāv iespēja, ka darbs tiks publicēts ”Baltic Journal of Modern Computing” (BJMC).

IZMANTOTIE INFORMĀCIJAS AVOTI

Abdar, Moloud, Farhad Pourpanah et al., "A review of uncertainty quantification in deep learning: Techniques, applications and challenges". *Information Fusion* 76 (2021. g. dec.), 243.—297. lpp. Pieejams: doi: 10.1016/j.inffus.2021.05.008. Pieejams: https://doi.org/10.1016/j.inffus.2021.05.008.

Ai, Qingzhong, Shiyu Liu et al., "Stein Variational Gradient Descent with Multiple Kernels". *Cognitive Computation* 15 (2021), 672.—682. lpp.

Avci, Mehmet Yigit, Ziyu Li et al., "Quantifying the uncertainty of neural networks using Monte Carlo dropout for deep learning based quantitative MRI". *ArXiv* abs/2112.01587 (2021).

Bernardo, José M & Adrian FM Smith. *Bayesian Theory*. John Wiley Sons, Ltd, 1994.

Blei, David M., Alp Kucukelbir et al., "Variational Inference: A Review for Statisticians". *Journal of the American Statistical Association* 112 (2016), 859.—877. lpp.

Blundell, Charles, Julien Cornebise et al., "Weight Uncertainty in Neural Networks". *ArXiv* abs/1505.05424 (2015).

Chen, Derek, Zhou Yu et al., "Clean or Annotate: How to Spend a Limited Data Collection Budget". *Proceedings of the Third Workshop on Deep Learning for Low-Resource Natural Language Processing* (2021).

Furnieles, Gabriel. *Sigmoid and SoftMax Functions in 5 minutes*. 2022. Pieejams: https://towardsdatascience.com/sigmoid-and-softmax-functions-in-5-minutes-f516c80ea1f9.

Gal, Yarin & Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". *ArXiv* abs/1506.02142 (2015).

Gal, Yarin, Jiri Hron et al., "Concrete Dropout". *NIPS*. 2017.

Gelman, Andrew, John B Carlin et al., *Bayesian Data Analysis*. CRC press, 2013.

González-Sopeña, J.M., V. Pakrashi et al., "An overview of performance evaluation metrics for short-term statistical wind power forecasting". *Renewable and Sustainable Energy Reviews* 138 (2021), 110515. lpp. ISSN: 1364-0321. Pieejams: doi: https://doi.org/10.1016/j.rser.2020.110515. Pieejams: https://www.sciencedirect.com/science/article/pii/S1364032120308005.

Harris, Charles R., K. Jarrod Millman et al., "Array programming with NumPy". *Nature* 585.7825 (2020. g. sept.), 357.—362. lpp. Pieejams: doi: 10.1038/s41586-020-2649-2. Pieejams: https://doi.org/10.1038/s41586-020-2649-2.

Hernández-Lobato, José Miguel & Ryan P. Adams. "Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks". *International Conference on Machine Learning*. 2015.

Hoffman, Matthew D. & David M. Blei. “Structured Stochastic Variational Inference”. *arXiv: Learning* (2014).

Hoffman, Matthew D., David M. Blei et al., “Stochastic variational inference”. *ArXiv* abs/1206.7051 (2012).

Hunter, J. D. “Matplotlib: A 2D graphics environment”. *Computing In Science & Engineering* 9.3 (2007), 90.—95. lpp.

Jespersen, Nicolai Schipper. “An Introduction to Markov Chain Monte Carlo”. *Research Methods & Methodology in Accounting eJournal* (2010).

Jordan, Michael I., Zoubin Ghahramani et al., “An Introduction to Variational Methods for Graphical Models”. *Machine Learning* 37 (1999), 183.—233. lpp.

Jospin, Laurent Valentin, Wray L. Buntine et al., “Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users”. *IEEE Computational Intelligence Magazine* 17 (2020), 29.—48. lpp.

Kendall, Alex & Yarin Gal. “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?”: *NIPS*. 2017.

Kingma, Diederik P., Tim Salimans et al., “Variational Dropout and the Local Reparameterization Trick”. *ArXiv* abs/1506.02557 (2015).

Kingma, Diederik P. & Max Welling. “Auto-Encoding Variational Bayes”. *CoRR* abs/1312.6114 (2013).

Kucukelbir, Alp, Rajesh Ranganath et al., “Automatic Variational Inference in Stan”. *NIPS*. 2015.

Lee, Kimin, Sukmin Yun et al., “Robust Inference via Generative Classifiers for Handling Noisy Labels”. *International Conference on Machine Learning*. 2019.

Lee, Sungyoon, Hoki Kim et al., “Graddiv: Adversarial robustness of randomized neural networks via gradient diversity regularization”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

Li, Junnan, Richard Socher et al., “DivideMix: Learning with Noisy Labels as Semi-supervised Learning”. *ArXiv* abs/2002.07394 (2020).

Lin, Zehui, Pengfei Liu et al., “DropAttention: A Regularization Method for Fully-Connected Self-Attention Networks”. *ArXiv* abs/1907.11065 (2019).

Meng, Xuhui, Hessam Babaee et al., “Multi-fidelity Bayesian Neural Networks: Algorithms and Applications”. *J. Comput. Phys.* 438 (2020), 110361. lpp.

Molchanov, Dmitry, Arsenii Ashukha et al., “Variational Dropout Sparsifies Deep Neural Networks”. *ArXiv* abs/1701.05369 (2017).

Nguyen, Son Minh, Duong Nguyen et al., “Structured Dropout Variational Inference for Bayesian Neural Networks”. *Neural Information Processing Systems*. 2021.

Nguyen, Trung Van & Edwin V. Bonilla. “Efficient Variational Inference for Gaussian Process Regression Networks”. *International Conference on Artificial Intelligence and Statistics*. 2013.

Nielsen, Michael A. *Neural networks and deep learning*. Determination Press, 2015.

Nixon, Jeremy, Michael W. Dusenberry et al., "Measuring Calibration in Deep Learning". *ArXiv* abs/1904.01685 (2019).

Olivier, Audrey, Michael D. Shields et al., "Bayesian neural networks for uncertainty quantification in data-driven materials modeling". *Computer Methods in Applied Mechanics and Engineering* 386 (2021), 114079. lpp.

Paszke, Adam, Sam Gross et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, 8024.—8035. lpp. Pieejams: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Ranganath, Rajesh, Sean Gerrish et al., "Black Box Variational Inference". *International Conference on Artificial Intelligence and Statistics*. 2013.

Reed, Scott E., Honglak Lee et al., "Training Deep Neural Networks on Noisy Labels with Bootstrapping". *CoRR* abs/1412.6596 (2014).

Salimans, Tim, Diederik P. Kingma et al., "Markov Chain Monte Carlo and Variational Inference: Bridging the Gap". *International Conference on Machine Learning*. 2014.

Salvatier, John, Thomas V. Wiecki et al., "Probabilistic Programming in Python using PyMC". *arXiv: Computation* (2015).

Shridhar, Kumar, Felix Laumann et al., "A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference". *ArXiv* abs/1901.02731 (2019).

— "Uncertainty Estimations by Softplus normalization in Bayesian Convolutional Neural Networks with Variational Inference". *arXiv: Learning* (2018).

Srivastava, Nitish, Geoffrey E Hinton et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research* 15 (2014).

Staber, Brian & Sébastien Da Veiga. "Benchmarking Bayesian neural networks and evaluation metrics for regression tasks". 2022.

Tötsch, Niklas & Daniel Hoffmann. "Classifier uncertainty: evidence, potential impact, and probabilistic treatment". *PeerJ Computer Science* 7 (2020).

Val. "100% Accurate Mushroom Classification in Python". *Medium* (2021). Pieejams: `https://val0.medium.com/100-accurate-mushroom-classification-in-python-eac61da3bace`.

Wang, Hao & Dit-Yan Yeung. "A Survey on Bayesian Deep Learning". *ACM Computing Surveys (CSUR)* 53 (2016), 1.—37. lpp.

Zhang, Chiyuan, Samy Bengio et al., "Understanding deep learning requires rethinking generalization". *ArXiv* abs/1611.03530 (2016).

Zhang, Yifan, Xueping Li et al., “Performance Comparison of Bayesian Deep Learning Model and Bayesian Shallow Neural Networks”. *Energies* 13.3 (2020), 648. lpp.

Zhou, Xiong, Xianming Liu et al., “Learning with Noisy Labels via Sparse Regularization”. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), 72.—81. lpp.

PIELIKUMI

1. Pielikumi

Tabula parāda visus ievāktos datus katram algoritmam, visu procentu jaukto klašu un iezīmju eksperimentos.

precision	recall	f1	accuracy	tp	fp	fn	tn	dispersija	std	algoritms	% jauktas klases	% jauktas iezīmes
95.34%	85.71%	90.27%	90.76%	696	34	116	778	0.105724	0.325152	VI	0%	0%
89.71%	86.95%	88.31%	88.49%	706	81	106	731	0.168235	0.410165	VI	10%	10%
94.47%	79.93%	86.59%	87.62%	649	38	163	774	0.201928	0.449364	VI	20%	20%
88.77%	81.77%	85.13%	85.71%	664	84	148	728	0.229843	0.479419	VI	30%	30%
82.63%	67.36%	74.22%	76.60%	547	115	265	697	0.242078	0.492015	VI	40%	40%
52.53%	24.26%	33.19%	51.17%	197	178	615	634	0.245685	0.495666	VI	50%	50%
100.00%	99.42%	99.71%	99.70%	807	0	5	812	0.002664	0.051611	BBB	0%	0%
99.29%	97.41%	98.34%	98.35%	791	6	21	806	0.003512	0.059261	BBB	10%	10%
98.48%	97.59%	98.02%	98.05%	792	12	20	800	0.003335	0.057746	BBB	20%	20%
99.49%	89.60%	94.23%	94.60%	728	4	84	808	0.0054	0.073486	BBB	30%	30%
98.33%	87.11%	92.32%	92.93%	709	12	103	800	0.01185	0.108856	BBB	40%	40%
56.37%	49.18%	52.24%	55.29%	399	310	413	502	0.012961	0.113846	BBB	50%	50%
100.00%	100.00%	100.00%	100.00%	812	0	0	812	1.07E-05	0.003275	MCD	0%	0%
100.00%	99.88%	99.94%	99.94%	811	0	1	812	0.000907	0.030111	MCD	10%	10%
99.82%	99.76%	99.79%	99.79%	810	1	2	811	0.002085	0.04566	MCD	20%	20%
98.23%	98.01%	98.12%	98.18%	797	14	15	798	0.0101	0.1005	MCD	30%	30%
86.52%	96.44%	91.18%	90.67%	782	123	30	689	0.039816	0.199539	MCD	40%	40%
60.40%	56.33%	58.17%	59.49%	457	305	355	507	0.073562	0.271223	MCD	50%	50%

2. Pielikums

Tabula parāda visus ievāktos datus katram algoritmam, visu procentu jaukto iezīmju eksperimentos.

precision	recall	f1	accuracy	tp	fp	fn	tn	dispersija	std	algoritms	% jauktas klases
94.59%	88.30%	91.34%	91.63%	717	41	95	771	0.109616	0.331083	VI	0%
91.83%	84.48%	88.01%	88.49%	686	61	126	751	0.124908	0.353423	VI	10%
92.05%	82.76%	87.16%	87.81%	672	58	140	754	0.12758	0.357183	VI	20%
88.55%	84.73%	86.60%	86.88%	688	89	124	723	0.138683	0.372402	VI	30%
90.92%	80.17%	85.21%	86.08%	651	65	161	747	0.140117	0.374322	VI	40%
86.14%	81.90%	83.96%	84.36%	665	107	147	705	0.146252	0.382429	VI	50%
100.00%	99.42%	99.71%	99.70%	807	0	5	812	0.002279	0.047741	BBB	0%
99.89%	98.19%	99.03%	99.04%	797	1	15	811	0.002896	0.05381	BBB	10%
99.42%	97.59%	98.49%	98.50%	792	5	20	807	0.002643	0.051412	BBB	20%
99.61%	91.65%	95.44%	95.68%	745	3	67	809	0.004186	0.064697	BBB	30%
99.33%	89.39%	94.05%	94.45%	727	5	85	807	0.00349	0.059073	BBB	40%
100.00%	84.47%	91.48%	92.33%	688	0	124	812	0.003237	0.056894	BBB	50%
100.00%	100.00%	100.00%	100.00%	812	0	0	812	1.52E-05	0.003901	MCD	0%
100.00%	99.89%	99.94%	99.94%	811	0	1	812	0.000239	0.015451	MCD	10%
100.00%	99.78%	99.89%	99.88%	810	0	2	812	0.000459	0.021417	MCD	20%
100.00%	99.78%	99.89%	99.88%	810	0	2	812	0.001031	0.032104	MCD	30%
100.00%	98.69%	99.33%	99.34%	801	0	11	812	0.002737	0.052318	MCD	40%
99.76%	91.66%	95.50%	95.77%	745	2	67	810	0.005411	0.073561	MCD	50%

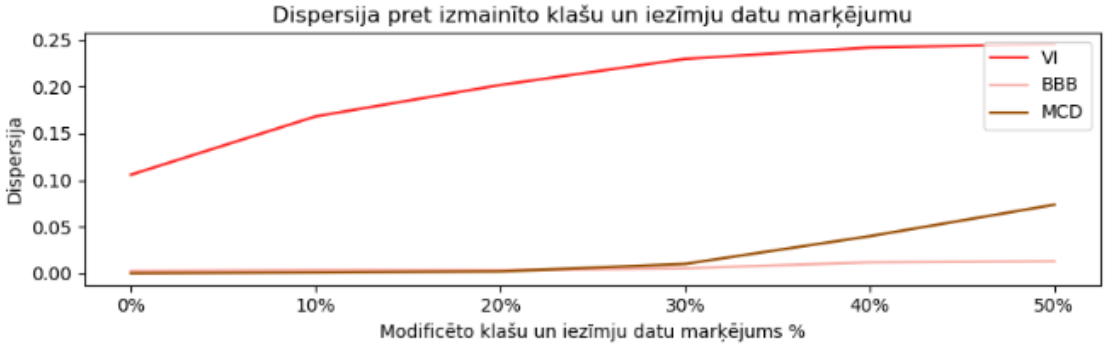
3. Pielikums

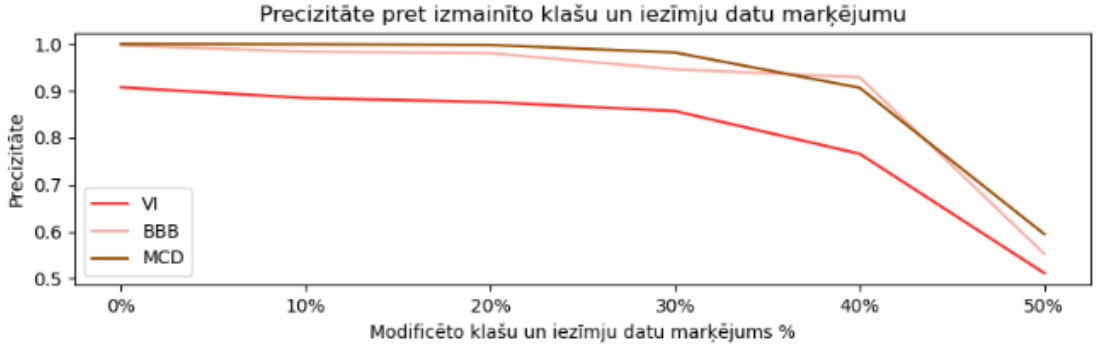
Tabula parāda visus ievāktos datus katram algoritmam, visu procentu jaukto klašu eksperimentos.

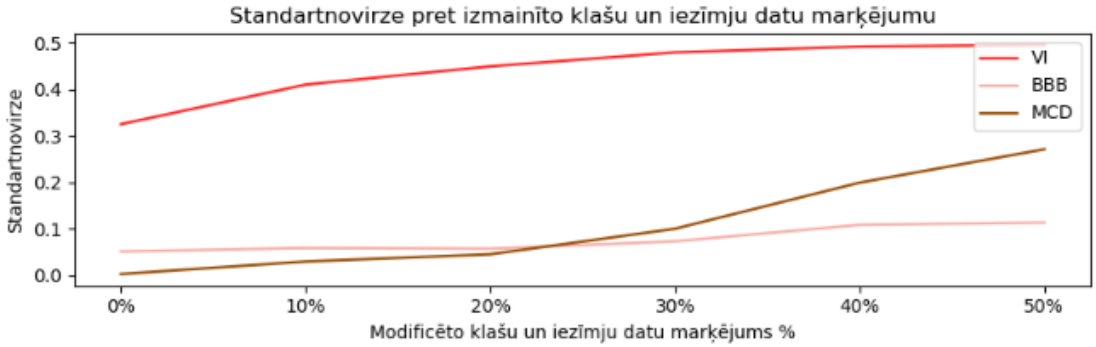
precision	recall	f1	accuracy	tp	fp	fn	tn	dispersija	std	algoritms	% jauktas klases
95.34%	85.71%	90.27%	90.76%	696	34	116	778	0.105724	0.325152	VI	0%
94.16%	89.29%	91.66%	91.87%	725	45	87	767	0.154754	0.393388	VI	10%
95.62%	83.37%	89.08%	89.78%	677	31	135	781	0.193729	0.440147	VI	20%
95.36%	86.08%	90.49%	90.95%	699	34	113	778	0.22615	0.475553	VI	30%
82.22%	86.58%	84.34%	83.93%	703	152	109	660	0.241911	0.491845	VI	40%
60.38%	19.70%	29.71%	53.39%	160	105	652	707	0.247039	0.49703	VI	50%
100.00%	99.66%	99.83%	99.82%	809	0	3	812	0.002348	0.04846	BBB	0%
100.00%	99.42%	99.71%	99.70%	807	0	5	812	0.002655	0.051527	BBB	10%
100.00%	99.42%	99.71%	99.70%	807	0	5	812	0.003957	0.062907	BBB	20%
99.16%	97.59%	98.36%	98.38%	792	7	20	805	0.007795	0.088289	BBB	30%
96.08%	97.85%	96.93%	96.91%	794	33	18	779	0.01371	0.117092	BBB	40%
57.13%	69.69%	60.86%	57.20%	566	453	246	359	0.00889	0.094286	BBB	50%
100.00%	100.00%	100.00%	100.00%	812	0	0	812	2.66E-05	0.005156	MCD	0%
100.00%	100.00%	100.00%	100.00%	812	0	0	812	0.000613	0.024751	MCD	10%
100.00%	99.88%	99.94%	99.94%	811	0	1	812	0.001219	0.034916	MCD	20%
98.30%	99.15%	98.72%	98.71%	805	14	7	798	0.006461	0.080381	MCD	30%
92.44%	97.57%	94.90%	94.72%	791	65	21	747	0.010166	0.100827	MCD	40%
50.29%	66.63%	57.23%	50.45%	540	533	272	279	0.007406	0.086056	MCD	50%

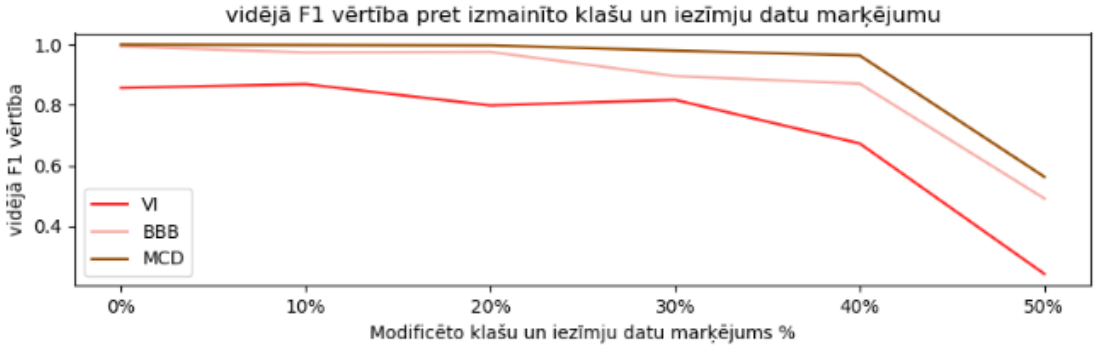
4. Pielikums

Grafiki, kas parāda dažādo rezultātu maiņu, visiem algoritmiem, mainoties klašu un iezīmju trokšņa daudzumam.



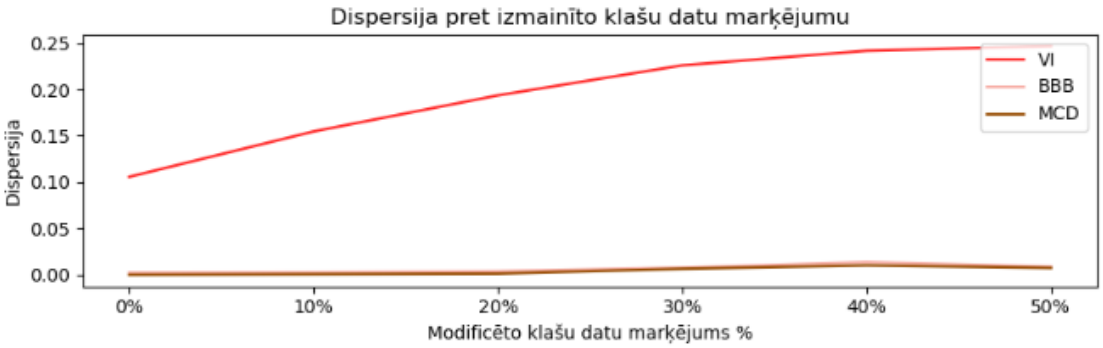


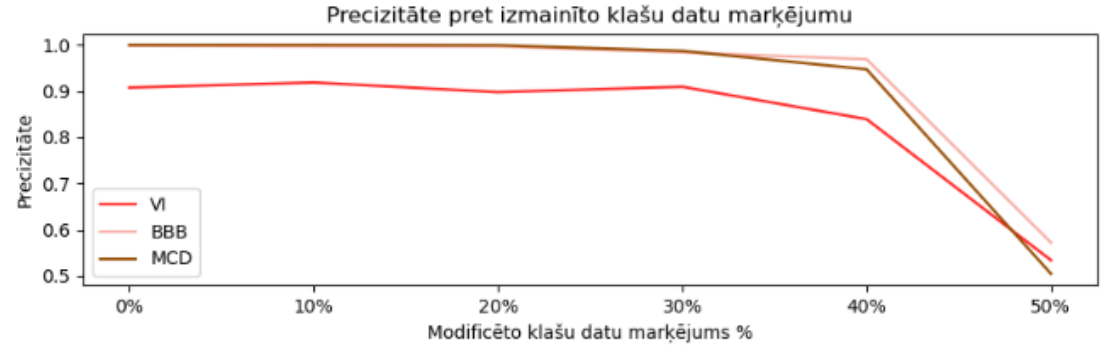


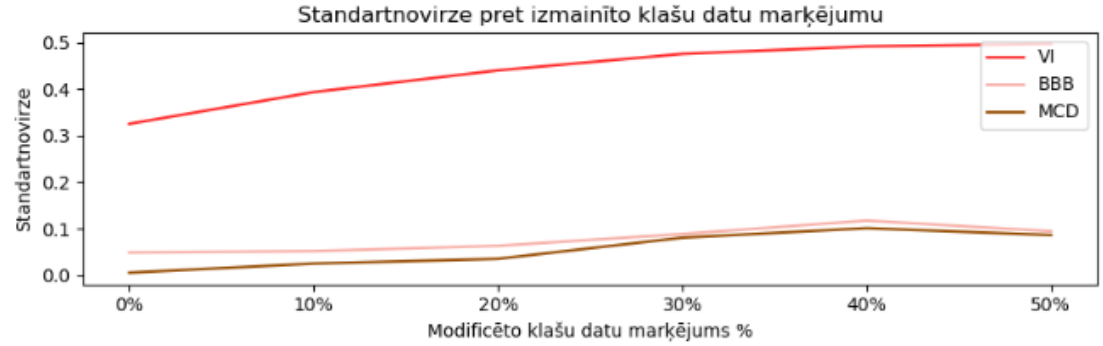


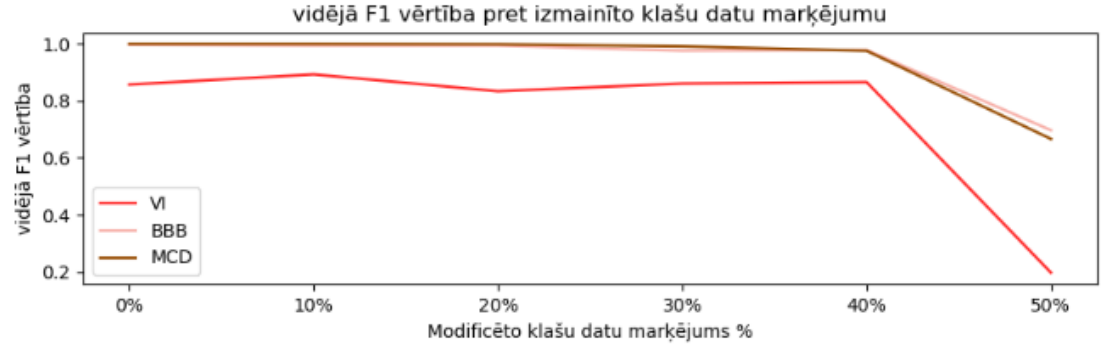
5. Pielikums

Grafiki, kas parāda dažādo rezultātu maiņu, visiem algoritmiem, mainoties klašu trokšņa daudzumam.



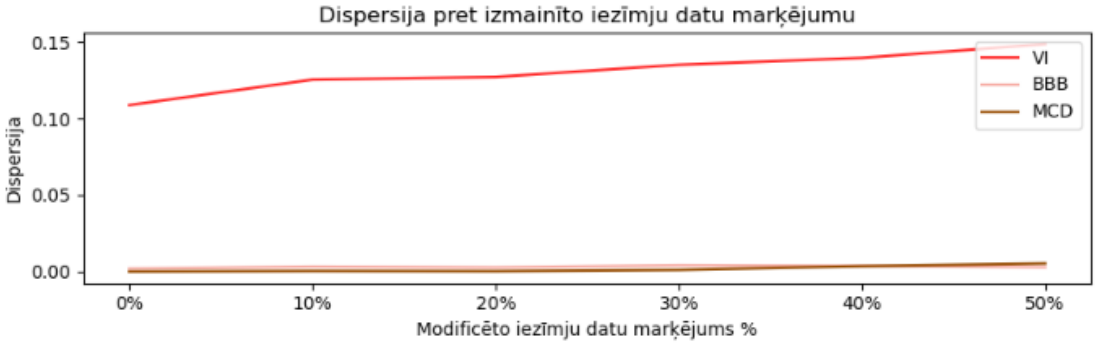


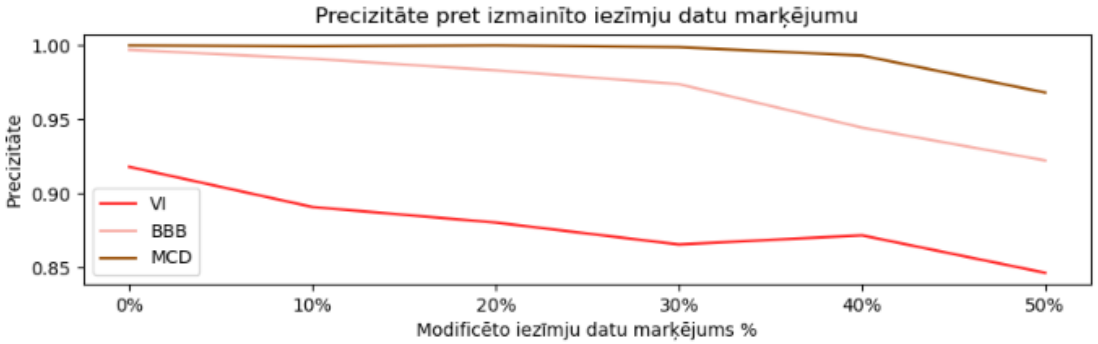


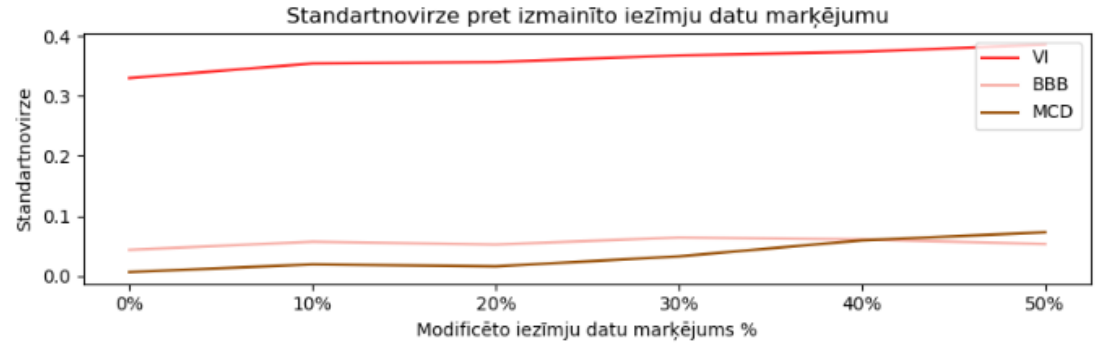


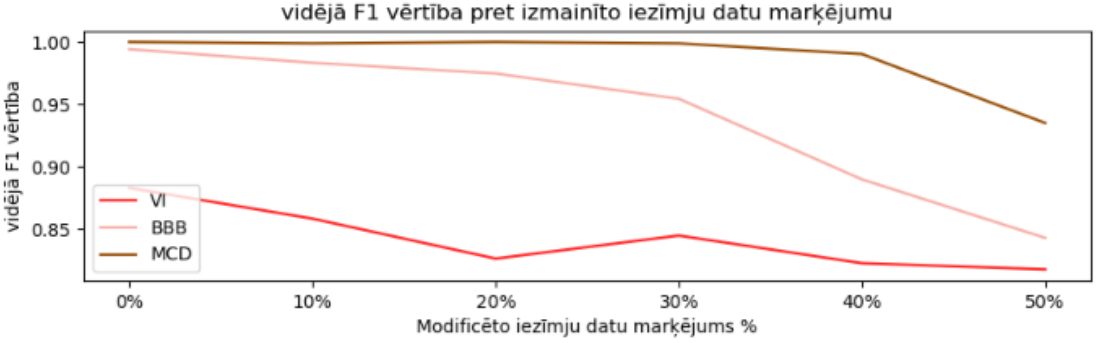
6. Pielikums

Grafiki, kas parāda dažādo rezultātu maiņu, visiem algoritmiem, mainoties iezīmju trokšņa daudzumam.



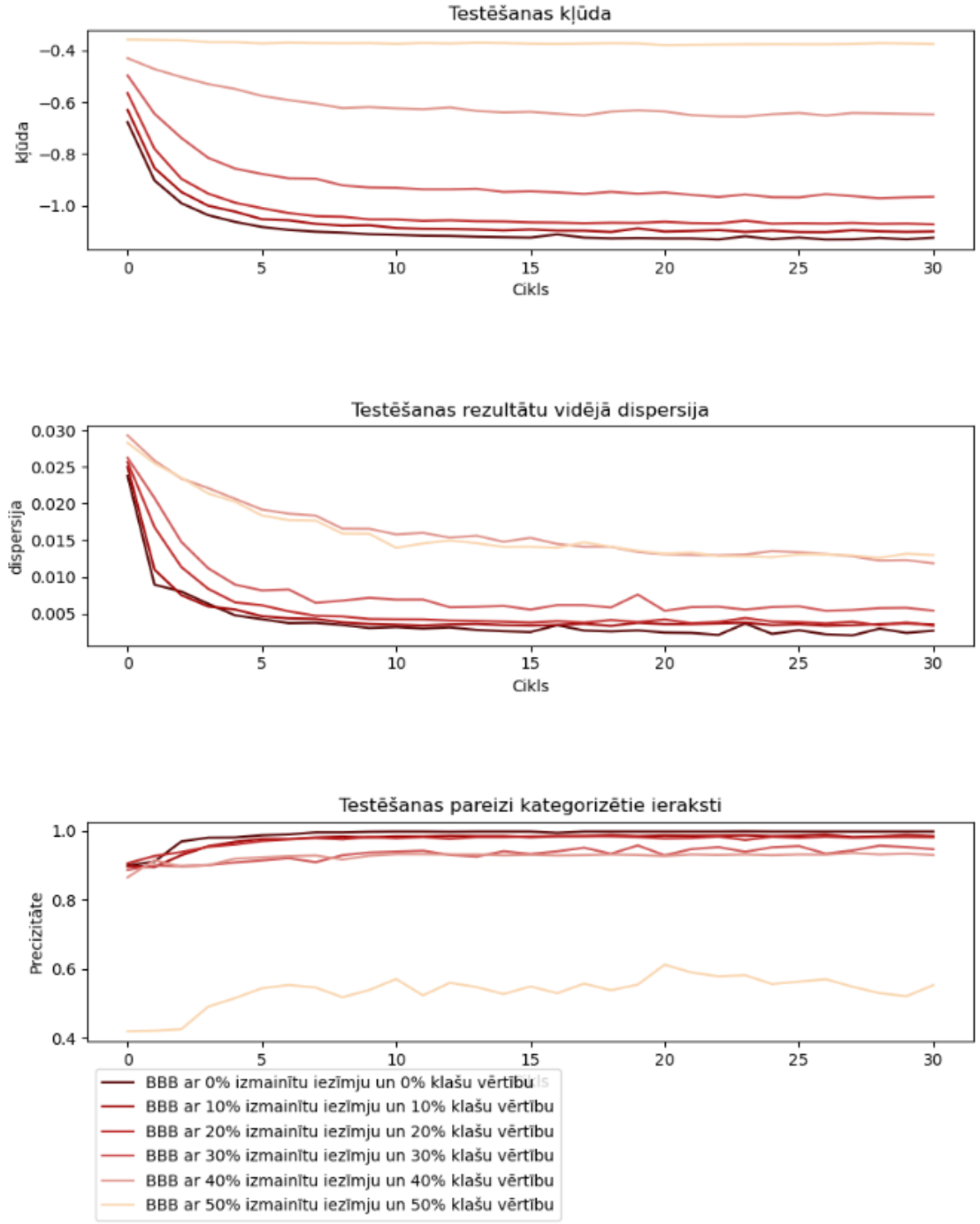






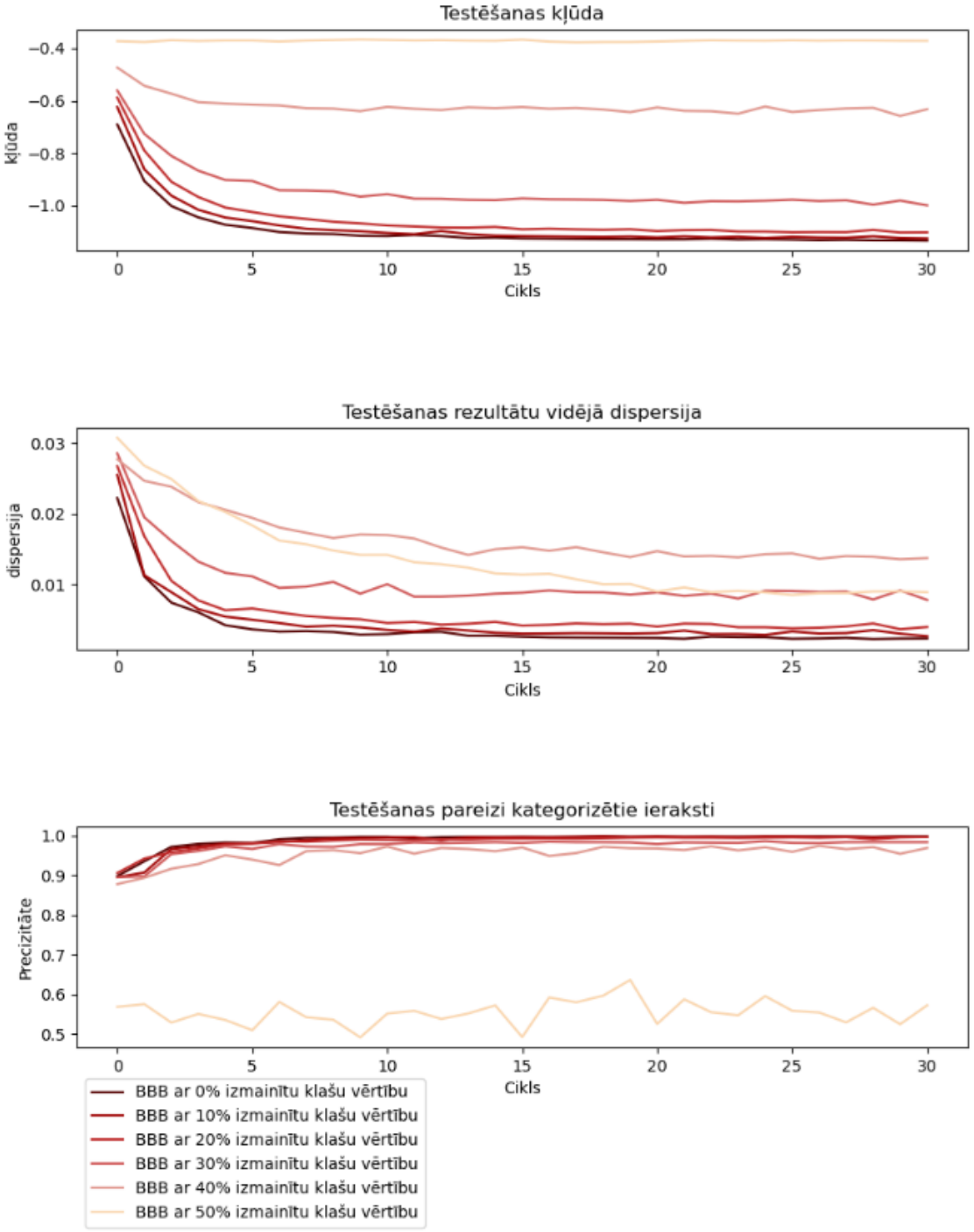
7. Pielikums

Grafiki, kas parāda ievākto rezultātu maiņu, BBB algoritmam, gadījumā, kad troksnis tika pievienots gan klasēm gan iezīmēm.



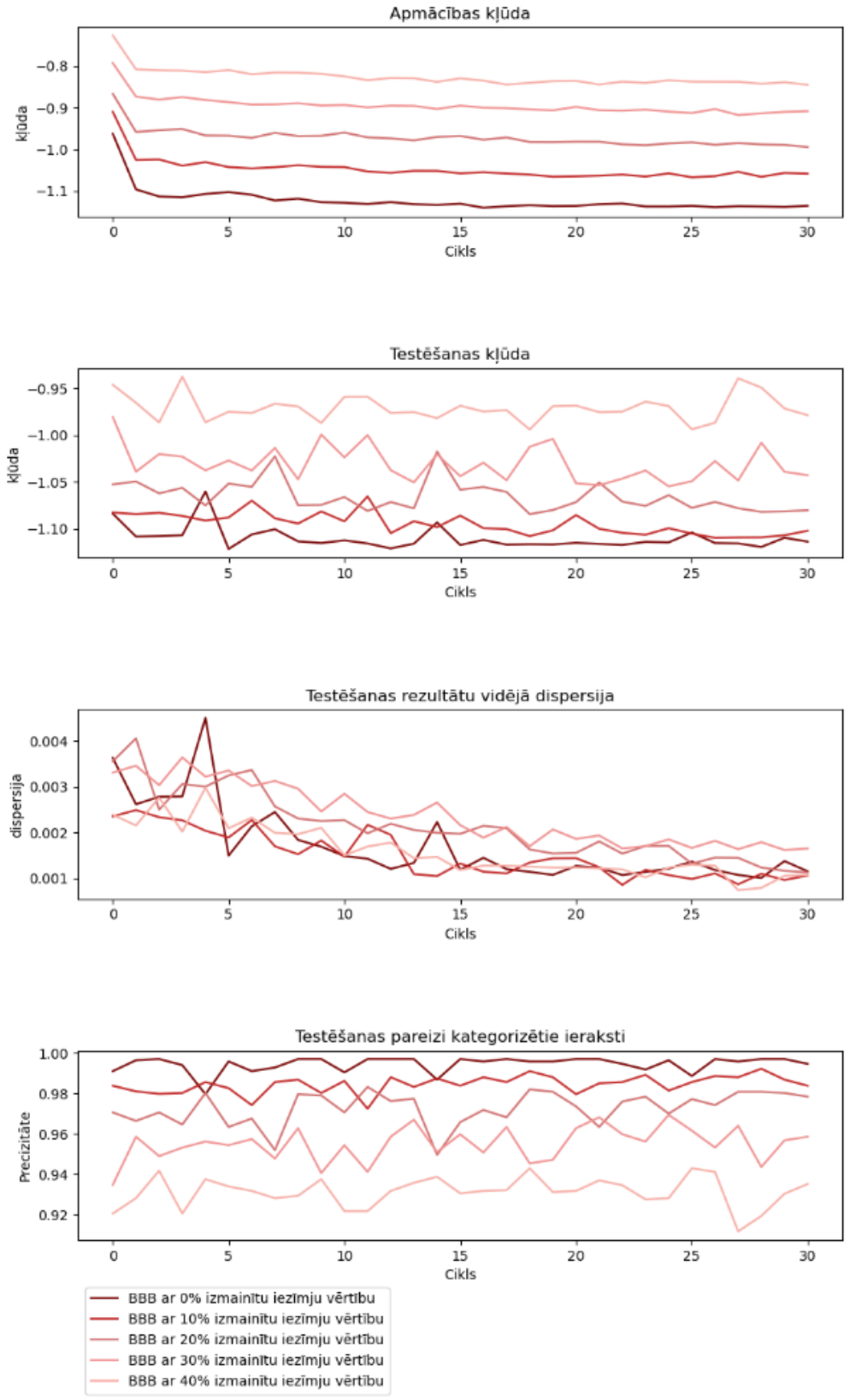
8. Pielikums

Grafiki, kas parāda ievākto rezultātu maiņu, BBB algoritmam, gadījumā, kad troksnis tika pievienots tikai klasēm.



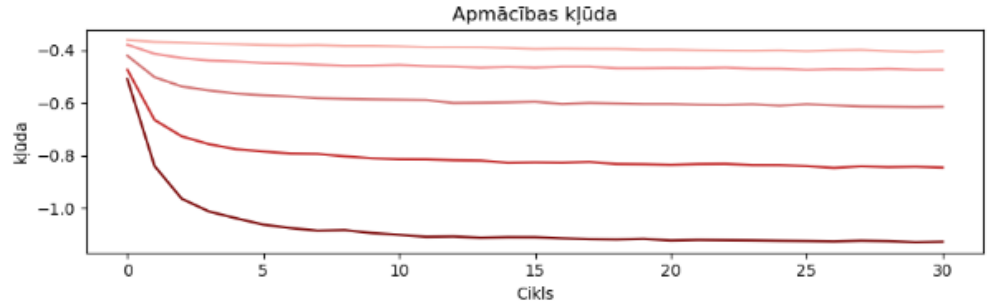
9. Pielikums

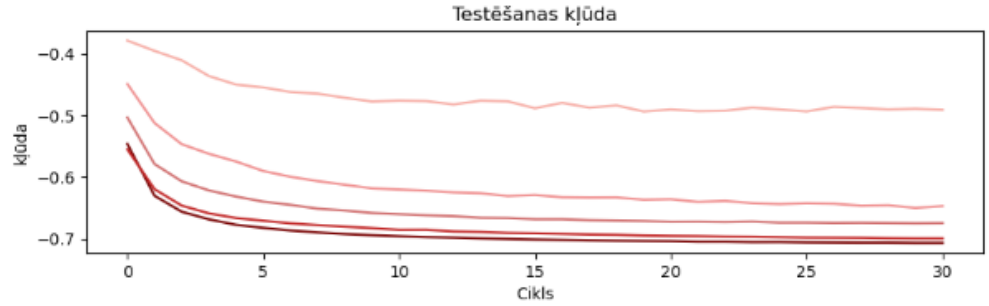
Grafiki, kas parāda ievākto rezultātu maiņu, BBB algoritmam, gadījumā, kad troksnis tika pievienots tikai iezīmēm.

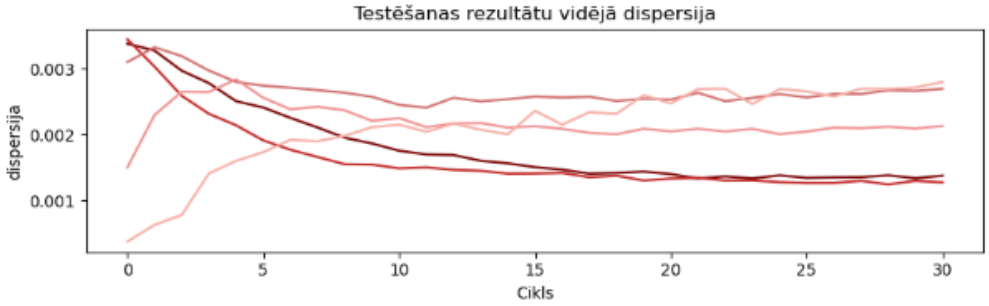


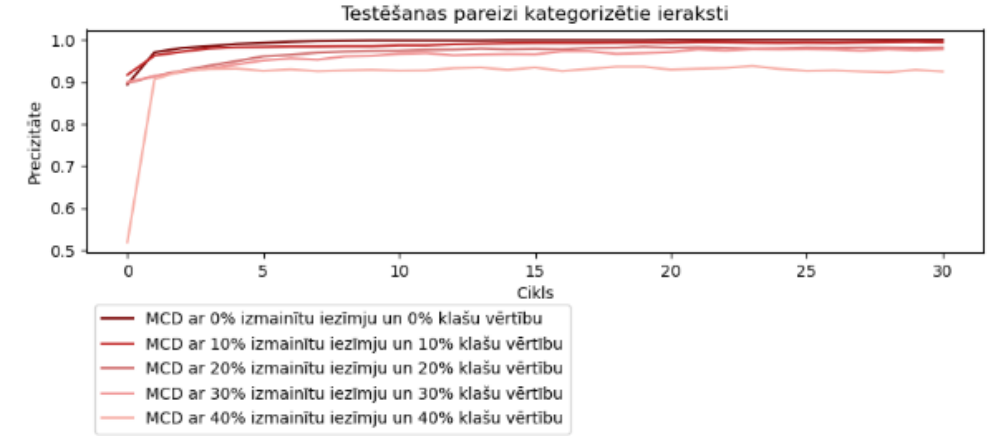
10. Pielikums

Grafiki, kas parāda ievākto rezultātu maiņu, MC-D algoritmam, gadījumā, kad troksnis tika pievienots gan klasēm gan iezīmēm.



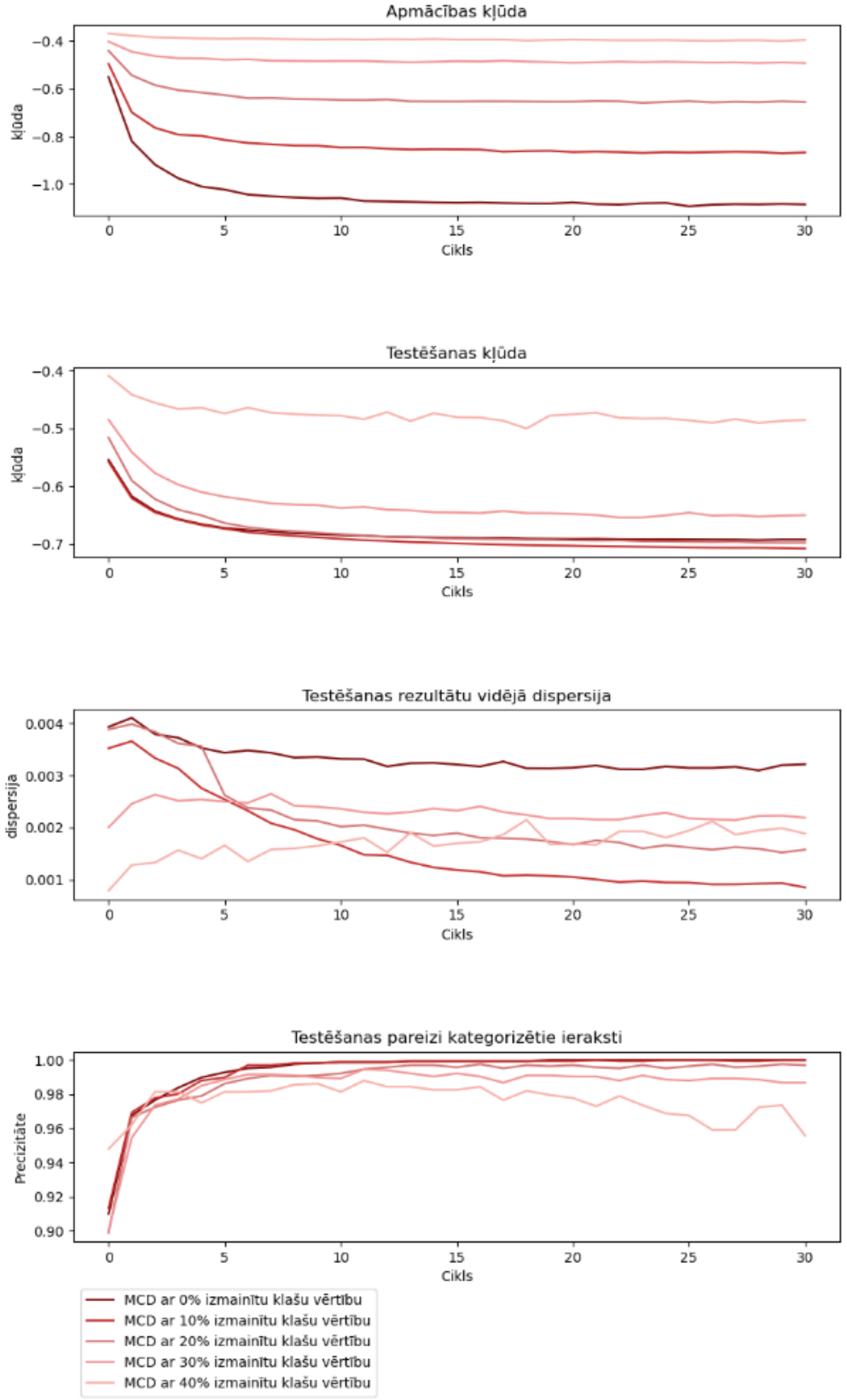






11. Pielikums

Grafiki, kas parāda ievākto rezultātu maiņu, MC-D algoritmam, gadījumā, kad troksnis tika pievienots tikai klasēm.



12. Pielikums

Grafiki, kas parāda ievākto rezultātu maiņu, MC-D algoritmam, gadījumā, kad troksnis tika pievienots tikai iezīmēm.