LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ATDARINOŠĀS MAŠĪNMĀCĪŠANĀS PIELIETOJUMS ROBOTIKĀ

MAĢISTRA KURSA DARBS

Autors: **Pēteris Račinskis**
Stud. apl. Nr. pr20015
Darba vadītājs: Dr. sc. comp. Modris Greitāns

RĪGA 2022

SATURS

1. IEVADS

Sacīt, ka mašīnmācīšanās šobrīd ir ļoti aktuāla pētniecības nozare, būtu maigi. Pēdējās desmitgades laikā tieši šis izpētes lauks ir eksplodējis popularitātē kā neviens cits, pateicoties galvenokārt diviem faktoriem: ļoti vispārīgiem neironu tīklu modeļiem un skaitļošanas resursu veiktspējai, kas beidzot ļāvusi šos teorētiski jau ļoti sen[1, 2, 3] iedomātos mākslīgā intelekta uzbūves elementus realizēt praksē. Tā risināti uzdevumi, ko izsenis daudzi uzskatījuši par neiespējamiem, un lietojuši kā argumentu pret mašīnmācīšanos kā rīku, kas spētu konkurēt ar bioloģiskas izcelsmes prātiem — semantiskas nozīmes meklēšana attēlos[4], tekstu korpusu analīze un ģenerēšana ar "izpratni" par to saturu[5] un visspējīgāko spēlētāju pārspēšana nepilnīgas informācijas spēlēs ar neaptverami milzīgiem iespējamo stāvokļu permutāciju skaitem[6].

Nav arī īpaši grūti atrast vēsturisko saikni starp mākslīgo intelektu un robotiku. Tautas iztēlē termins "robots" drīzāk droši vien iezīmēs zinātniskās fantastikas radītos personāžus — mehāniskas būtnes, kas spēj patstāvīgi darboties neierobežotā vidē un risināt sarežģītus uzdevumus — nevis pieticīgākus, reāli pastāvošus un ražotnēs rodamus industriālos robotus. Un šī pati zinātniskā fantastika radījusi arī nesaraujamu saiti starp robotiem un mākslīgo intelektu[7] — diskusijas par mākslīgo intelektu bieži plūstoši pāriet diskusijās par ar šādu intelektu aprīkotiem robotiem, un šo robotu neizbēgami kareivīgajām ambīcijām attiecībā pret cilvēci. Protams, zinātne ne vienmēr seko populārzinātniskās iedomas lidojumam, taču šāda saikne ir visnotaļ pamatota — spēja mācīties no paraugiem vai patstāvīgi un pielāgoties savai apkārtnei ir ārkārtīgi noderīga, jo daudzi uzdevumi, kuru risināšanai varētu pielietot robotus, ir sarežģīti nevis to fizikālajā izpildē, bet tieši vadības uzdevuma formulēšanā un realizācijā.

Atdarinošā mašīnmācīšanās (*imitation learning*) ir viens no paņēmieniem, ar kuriem tiek mēģināts risināt šādas sarežģītas vadības problēmas. Lai gan pamatu pamatos nevar apgalvot, ka tā ir tikai robotikai piemērota metožu saime, lielākā daļa izpētes virzīta tieši šajā virzienā — problēmas tiek formulētas kā fizikālu (vai nosacīti fizikālu — virtuālās vidēs simulētu) procesu kontroles uzdevumi, un risinājumi tiek rasti no pēc iespējas mazāka skaita veiksmīgas darbības piemēru. Mašīnmācīšanās nozarē bioloģiskas analoģijas un iedvesma nav nekāds retums, un savā ziņā šāda mācīšanās atspoguļo vienu no izplatītiem paņēmieniem, kā cilvēki vai sabiedriski dzīvnieki nodod prasmes viens otram - demonstrējot. Nevar nepieminēt, ka izpēte šajā jomā bieži aizņemas pieejas un iespaidojas no rezultātiem, kas gūti ar stimulēto mašīnmācīšanos (*reinforcement learning*) - savā ziņā vispārīgu, pašmācībai un treniņam analoģisku paņēmienu. Arī abu metožu apvienojums ir ideja, kas pavīd visai regulāri — cerībā, ka, atdarinot ekspertus, var ātrāk nonākt pie derīgām stratēģijām, kas var kalpot kā sākumpunkts dziļākai pašmācībai; vai arī izmantot šādu stimulēto metodi, lai precīzāk imitētu treniņa datus.

1.1. Darba mērķis un struktūra

Šis ir maģistra kursa darbs - pirmais konkrētais rezultāts, kas sasniegts maģistra darba izstrādes procesā. Tāpēc ir jārēķinās ar diezgan īpatnēju formātu un saturu - tiek dokumentēta kāda pētnieciska projekta pirmā fāze, kas bieži vien sastāv no dažādu literatūras avotu izpētes un personiskiem treniņiem, vēl pirms iespējams nopietni sākt eksperimentālu darbību vai pat izvirzīts konkrēts mērķis visam projektam.

Arī šis gadījums nav nekāds izņēmums. Sākumā izvēlēta ļoti aptuvena tēma, balstoties uz Elektronikas un datorzinātņu institūta ekspertu ieteikumiem, un pirmajā darba semestrī lielākoties veikta attiecīgās nozares apguve pašmācības ceļā. Šī nodarbe sastāvējusi galvenokārt no divu veidu darbībām — zinātniskās literatūras lasīšanas un tajā aprakstīto teorētisko jēdzienu un praktisko metožu apguves ar vienkāršiem eksperimentiem personiskās izpratnes veicināšanai.

Līdz ar to šīs atskaites galvenais mērķis ir sniegt ieskatu līdz šim maģistra darba gatavošans ietvaros paveiktajā un apgūtajā. Tā sastāv no trim galvenajām daļām:

1) ievada, kurā īsi izklāstīti vispārīgi jēdzieni, kas nepieciešami, lai izprastu zinātnisko literatūru nozarē;
2) pētniecisku rakstu izlases iztirzājuma un salīdzinājuma;
3) neliela apraksta par paša veikto darbību, apgūstot mašīnmācīšanās modeļus un to realizācijai nepieciešamo programnodrošinājumu.

1.2. Terminoloģijas tulkojumi

Viena no īpatnībām, ar ko ir nācies saskarties, strādājot tieši ar mašīnmācīšanās nozari, ir nepārprotamas terminoloģijas trūkums latviešu valodā. Pati zinātnes nozare, lai arī nebūt ne tik jauna kopumā, piedzīvojusi milzīgas izmaiņas un nepieredzētu uzplaukumu pēdējās desmitgades laikā. Protams, datorzinātnes laukā pirmā un galvenā saziņas valoda ir angļu. Attiecīgi novērajami divējādi un saistīti fenomeni - publikācijas un terminoloģija, kas radītas senāk, veidojušas dziļi specifisku nišu, kas nav iedvesmojusi daudz mēģinājumu tulkot to uz citām valodām, savukārt uzplaukuma laikos vēl ir ļoti daudz materiāla, ko vienkārši neviens nav paguvis iztulkot.

Patvaļīgi izvēloties tulkojumu, pastāv risks mulsināt lasītāju un sadrumstalot jau tā nelielo literatūras kopu dažādu atslēgas vārdu izvēles rezultātā. Tāpēc šeit izveidots saraksts ar potenciāli mulsinoši tulkoto terminoloģiju tās oriģinālajā formulējumā angļu valodā, izvēlētajiem tulkojumiem un īsiem pamatojumiem.

1) *policy* — **stratēģija**. Šis termins pamatā tiek lietots, lai apraksttītu kādu funkciju, kas novērojumus attēlo lēmumu telpā. Pirmais ieraksts tieši tāpēc, ka varētu būt strīdīgākais. Angļu valodā pastāv divi termini, *policy* un *politics*, kas parasti latviski tiek tulkoti vienādi — politika — par spīti radikāli atšķirīgām nozīmēm. Termins *strategy* tiek lietots kā sinonīms pirmajam abās valodās, un arī piemērojams tieši šādām lēmumu pieņemšanas funkcijām, piemēram, spēļu teorijā.
2) *reinforcement learning* — **stimulētā mašīnmācīšanās**. Meklējumi tiešsaistē atklāj

[8], ka šis tulkojums jau ir samērā izplatīts, taču varētu būt nezināms lasītājiem, kas ar to sastopas pirmo reizi — pat ja zināms metodes angliskais nosaukums.

3) *imitation learning* — **atdarinošā mašīnmācīšanās**. Paša autora piedāvāts tulkojums, izmantojot iepriekšējo kā piemēru, jo nav izdevies atrast alternatīvas. Latviskais vārds "atdarināt" izvēlēts pār internacionālismu "imitēt", jo to vieglāk izlocīt formā, kas neizklausās lauzīta un neveikla. Taču procesā zūd spēja viegli atrast sākotnējo vārdu svešvalodā, kas ļoti svarīga zinātniskajā vidē, kurā latviski pieejamo resursu ir maz.

1.3. Tehniskās priekšzināšanas, definīcijas

Pētot un veidojot spriedumus par zinātnisko literatūru viens no lielākajiem šķēršļiem lasītājam "no malas" ir katrā nozarē pieņemtais tehnisko priekšzināšanu kopums, ko autori sagaida no auditorijas. Tas, protams, ir loģiski, jo publikācija, kas apraksta jaunākos atklājumus kādā dziļi specifiskā lauciņā, nevar veltīt visu sev atvēlēto drukas apjomu elementāras un vispārzināmas terminoloģijas skaidrojumiem. Tāpat, tālāk atskaitē iztirzājot šos rakstus, noderīgi ir ieviest tiem kopīgus apzīmējumus un definēt visus vienuviet.

1.3.1. Parametriski modeļi, šabloni

Viens no visplašāk izmantotajiem formālismiem datizraces un mašīnmācīšanās laukos ir parametriskais modelis. Pamatā tam ir ideja, ka nezināmu funkciju, kuras rezultātus vēlamies paredzēt, var aproksimēt ar citu funkciju jeb modeli:

$$M(x) \approx f(x) \tag{1.1}$$

Protams, šādu modeļu varētu būt bezgalīgi daudz, un tie visi var atšķirties pēc tā, cik labi spēj paredzēt nezināmās funkcijas vērtības. Tāpēc modeļu meklēšanai parasti izmanto šablonus - funkcijas, kuru argumentā papildus ievades datiem ir brīvi maināmi un kopīgi (tātad "apmācāmi") parametri θ:

$$\text{Meklē } \theta : M(x|\theta) = M_\theta(x) \approx f(x) \tag{1.2}$$

Iegūtā šablona funkcijas un apmācīto parametru kombinācija $\{M, \theta\}$ tad veido konkrētu modeli. Labs šablons ir tāds, kas spēj pielāgoties ļoti daudzām dažādām funkcijām:

$$\forall f \forall x \exists \theta : M_\theta(x) \approx f(x) \tag{1.3}$$

Atkarībā no uzdevuma specifikas, izplatīti modeļi mēdz būt regresori, kas aproksimē (parasti vektoriālas) funkcijas ar skaitliskām vērtībām,

$$f : x \to \mathbb{R}^k \tag{1.4}$$

$$M : x \times \theta \to \mathbb{R}^k \tag{1.5}$$

un klasifikatori, kas paredz ievades datu punkta piederību kādai diskrētai klasei

$$f : x \to C = \{c_1, c_2, ..., c_m\} \tag{1.6}$$

$$M : x \times \theta \to C \tag{1.7}$$

Bieži vien noderīgi ir ne tikai spēt attēlot datu punktu kā diskrētu klasi, bet iegūt varbūtību sadalījumu, kas apraksta tā iespējamību piederēt jebkurai no klasēm:

$$M : x \times \theta \times c_i \to [0; 1] \tag{1.8}$$

$$M_\theta(x, c_i) = P_i \tag{1.9}$$

$$\sum_{i=1}^{m} P_i = 1 \tag{1.10}$$

Lai varētu novērtēt, cik labi modelis aproksimē nezināmo funkciju, un vadīt parametru apmācības procesu, tiek izmantotas mērķa funkcijas (*loss functions*)[9]:

$$\ell : M_\theta(x) \times f(x) \to \mathbb{R} \tag{1.11}$$

Strādājot ar reāliem datiem, datu punkti veido datu kopu, kas parasti tiek uzskatīta par gadījuma izlasi no punktus ģenerējošā varbūtību sadalījuma. Praktiskiem apmācības uzdevumiem datu kopa parasti jāiegūst formā, kas satur gan sagaidāmos ievades datus, gan pareizu rezultātu:

$$s \sim \mathcal{D} \Leftrightarrow s \text{ ir no varbūtību sadalījuma } \mathcal{D} \tag{1.12}$$

$$y_i = f(x_i) \tag{1.13}$$

$$s_i = (x_i, y_i) \tag{1.14}$$

$$S = \{s_1, s_2, ..., s_n | s_i \sim \mathcal{D}\} \tag{1.15}$$

Datu kopai var aprēķināt empīrisku mērķa funkcijas novērtējumu,

$$L_S(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(M_\theta(x_i), y_i) \tag{1.16}$$

bet apmācības process parasti kādā veidā tiecas minimizēt šīs vērtības matemātisko cerību ģenerējošam sadalījumam (nevis tikai pašai datu kopai - ja modelis ļoti cieši pielāgots konkrētai datu izlasei bet zaudē precizitāti sadalījumam kopumā, to sauc par pārpielāgošanos — *overfitting*)

$$L_\mathcal{D}(\theta) = \mathbb{E}_\mathcal{D}[\ell(M_\theta(x_i), y_i)] \tag{1.17}$$

$$\text{Apmāca } M_\theta \text{ uz } \mathcal{D} \to \text{Minimizē } L_\mathcal{D}(\theta) \tag{1.18}$$

Ja modelis ir stratēģija (*policy*), stimulētās vai atdarinošās mašīnmācīšanās literatūrā to ļoti bieži izsaka kā $\pi_\theta(x)$. Mazliet mulsinošs ir tieši ar imitējošām metodēm saistītos rakstos lietotais apzīmējums π^*, ar ko apzīmē t.s. "ekspertu stratēģijas" — kas pašas ir nezināmās funkcijas, ko cenšamies aproksimēt pēc to ģenerēto punktu kopām.

1.3.2. Neironu tīkli

Neironu tīkls ir izplatīta modeļu šablonu saime, ko var izmantot dažādas formas funkciju aproksimēšanai — tie var būt gan klasifikatori, gan regresori, un pastāv ļoti dažādas to uzbūves variācijas, kas daļēji teorētiski, daļēji empīriskas eksperimentācijas rezultātā un daļēji kopējot bioloģiskās sistēmās atrodamas struktūras izstrādātas dažādu uzdevumu veikšanai. Neironu tīklu kopīgais elements ir t.s. perceptorns, kas izteikts jau pašos pirmsākumos[1]. Perceptrons funkcija, kas piemēro nelineāru aktviācijas funkciju σ argumentu vektora $\vec{x}$ elementu savstarpējai lineārai kombinācijai, t.i,

$$f_{perceptron}(\vec{x}) = \sigma(\vec{w} \cdot \vec{x} + b) \tag{1.19}$$

kur $\vec{w}$ ir t.s. svaru vektors, bet b — nobīde. Perceptrona parametri tātad ir brīvie mainīgie $\vec{w}$ un b. Neironu tīkls parasti sastāv no slāņiem — perceptronu f_i kopām, kas visi apstrādā to pašu argumentu vektoru, bet katrs ar saviem parametriem $\vec{w}_i, b_i$. Tad slāni algebriski izsaka formā

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ ... \\ w_k^T \end{bmatrix}; \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_k \end{bmatrix}; \tag{1.20}$$

$$f_{layer}(\vec{x}) = \sigma(W\vec{x} + \vec{b}) \tag{1.21}$$

Ja slānis tīklā ir pēdējais un tā vērtības ir modeļa izvadē, to sauc par izvades (*output*) slāni. Ievades datu vektoru sauc par ievades (*input*) slāni. Pārējos slāņus sauc par slēptajiem (*hidden layers*). Saka, ka slāņi savā starpā pilnīgi savienoti (*fully connected*), ja katram viena slāņa perceptronam argumentā parādās visi iepriekšējā slāņa izvades elementi. Svarīga neironu tīkla īpašība — ja tā aktivācijas funkcijas ir diferencējamas, tad arī tīkls kopumā ir diferencējams pēc katra tā parametra, pat ar perceptroniem daudzos slāņos. Līdz ar to var izmantot t.s. *backpropagation* algoritmu, kas atrod mērķa funkcijas parciālos atvasinājumus pēc modeļa parametriem un izmanto kādu gradientu optimizācijas metodi apmācībai.

Pastāv dažādas šo tīklu arhitektūras. Vienkāršākās sastāv no viena vai vairākiem slāņiem (neskaitot ievades slāni), taču ir plaši izplatīti arī, piemēram, konvolucionālie neironu tīkli[4], ko izmanto attēlu apstrādē, tai skaitā šajā atskaitē aplūkotajos pētījumos, kur nepieciešams gūt informāciju no video datiem. Galvenā atšķirība konvolucionālajā tīklā ir t.s. kodola funkciju jeb kerneļu (*kernel*) izmantošana - konvolucionāli slāņi vienā līmenī piemēro identiskas perceptrona funkcijas nelieliem iepriekšējā slāņa (matricas vai tenzora formā) reģioniem. Tas palīdz identificēt dažādas lokālas struktūras, piemēram, attēlā. Šo un vēl citu veidu sarežģītāku neironu tīklu arhitektūra ir ļoti plašs lauks, ko detalizēti šeit iztirzāt nav iespējams.

1.3.3. Markova lēmumu procesi

Pastāv dažādi formālismi procesu definēšanai vadības sistēmu izstrādes mērķiem, lai ar tiem varētu veikt matemātiskas operācijas. Izplatīti atdarinošās un stimulētās

mašīnmācīšanās literatūrā ir Markova lēmumu procesi (MDP — *Markov decision processes*), kas izmantojami situācijās, kad sistēmas stāvokli nākotnē pilnībā nosaka pašreizējais. Dažādi autori, kas darbojas dažādos izpētes virzienos, mēdz piedāvāt dažādus tā formulējumus, taču parasti tie ir ekvivalenti sekojošam[10]

$$MDP = (S, A, R, T, \gamma) \tag{1.22}$$

kur S — sistēmas iespējamo stāvokļu s kopa; A — kontrolētajam procesam ("aģentam") pieejamo darbību a kopa; $R : S \times A \to \mathbb{R}$ vai $R : S \to \mathbb{R}$— atdeves (*reward*) funkcija, kas ļauj kārtot sasniegtos stāvokļus pēc to tīkamības; $T : S \times A \to S$ vai $P(s' \in S)$ — pārejas (*transition*) funkcija, kas nosaka nākamo stāvokli s' vai tam atbilstošu varbūtību sadalījumu, ja pie iepriekšējā stāvokļa s izvēlēta darbība a; γ — koeficients nākotnes atdevju vērtību samazināšanai. MDP ir *galīgs* ja S, A ir galīgas kopas. Ja $s' = T(s, a)$ ir determinēts, MDP ir determinēts. Ja s' ir gadījuma lielums, kas pieder sadalījumam $P(s') = T(s, a)$, MDP ir stohastisks.

Atdarinošās mašīnmācīšanās metodēm ne vienmēr ir nepieciešams definēt atdeves funkciju un attiecīgi arī γ, taču tie ir nepieciešami metodēm, kas lieto stimulēto mašīnmācīšanos. Tā kā parasti spriests tiek par stratēģijām π_θ, kas izvēlas nākamo darbību a atkarībā no sistēmas stāvokļa s, tad bieži vien faktiskā pārejas funkcija ir formā $P(s') = T(s, \pi_\theta(s), s')$, t.i., pārejas funkcija apraksta "vides" (*environment*) reakciju uz aģenta (modeļa, stratēģijas) darbību. Ļoti izplatītas ir arī situācijas, kad modelis ņem vērā nevis pilno sistēmas stāvokli, bet gan t.s. novērojumu (*observation*) — $\pi_\theta(o) = \pi_\theta(g(s))$. Tā ir funkcija no kādas stāvokli raksturojošo parametru apakškopas, un bieži vien ļoti nepilnīgi šo stāvokli raksturo.

Trajektoriju, kādai process seko ar laika soļiem $t = \{1, 2, ..., T\}$, raksturo laikrinda (*state-action*) pāru formā — $((s_1, a_1), (s_2, a_2), ..., (s_T, a_T))$. Stāvokļus tajā, protams, iespējams aizstāt ar novērojumiem situācijās, kad tiek izmantota nepilnīga informācija. Ne vienmēr vēlams vai iespējams modelēt sistēmu ar MDP. Ir iespējami gadījumi, kad pārejas funkcija vai stratēģija ir atkarīga no laika soļa, kā arī sistēmas, kurās ar novērojumiem nepietiek lēmuma pieņemšanai un nepieciešams ņemt vērā iepriekšējo stāvokļu un darbību virkni, lai pareizi spriestu par slēptiem stāvokļa atribūtiem.

1.3.4. Stimulētā mašīnmācīšanās

Stimulētā mašīnmācīšanās ir pati par sevi ļoti aktuāla izpētes nozare, un nereti nodarbojas ar to pašu vai līdzīgu uzdevumu risināšanu, kā atdarinošā. Pastāv ne tikai kombinēti paņēmieni[11, 12], bet arī atdarināšanas metodes, kas tiešā veidā izmanto stimulēto mācīšanos, lai atdarinātu trajektoriju demonstrācijas[13]. Tāpēc nav nekāds pārsteigums, ka šis termins visnotaļ bieži parādās ar atdarinošo mašīnmācīšanos saistītos pētījumos, citreiz bez nekādiem papildus paskaidrojumiem.

Stimulētās mašīnmācīšanās teorētiskie pamati ir galīgi MDP un Belmana vienādojums[14]. Pieņem, ka katram stāvoklim ir kāda atdeve $R(s_t)$, bet uzdevums — maksimizēt šo atdevju summu visā trajektorijas garumā $\sum_{t=1}^{T} R(s_t)$. Tad var izteikt arī varbūtību sadalījumu atdevei katram stāvokļa un darbības pārim

$$p(s', r|s_t, a_t) = P[s_{t+1} = s', r = R(s')] \tag{1.23}$$

Nākotnē sagaidāmās atdeves vērtības, ņemot vērā dilšanas koeficientu γ, var izteikt kā

$$G_t = \sum_{k=0}^{T-t} \gamma^k R(s_{t+k+1}) \tag{1.24}$$

Jebkura stratēģija katram stāvoklim nosaka darbību vai darbību sadalījumu $p(a|s) = \pi(a, s)$. Var izmantot rekursīvu sakarību, lai katram stāvoklim piekārtotu sagaidāmo atdevi jeb vērtību $v_\pi(s)$, kas atkarīga no izmantotās stratēģijas — Belmana vienādojumu.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \sum_a \pi(a, s) \sum_{s',r} p(s', r|s_t, a_t)[r + \gamma v_\pi(s')] \tag{1.25}$$

Atrisināt mācīšanās uzdevumu tādā gadījumā nozīmē atrast stratēģiju, kas maksimizē atdevi. Pastāv dažādas metodes, kā to darīt. Teorētiski vienkāršākais taču praktiskiem uzdevumiem reti piemērojams paņēmiens ir tā saucamā Q-mācīšanās. Tā strādā samērā vienkārši — tiek izveidots tenzors Q ar elementu, kas atbilst katrai iespējamai (s, a) vērtībai, tam tiek piešķirta kāda sākotnējā vērtība (piemēram, 0).

Apmācība notiek, izvēloties

$$a_t = \max_a(Q(s_t = s)) \tag{1.26}$$

un sasniedzot trajektorijas beigas — vai nu pēc noteikta soļu skaita T, vai arī kāda pārtraukšanas nosacījuma. Tad iegūtajai trajektorijai $(s_1, a_1), (s_2, a_2), ..., (s_T, a_T)$ no beigām aprēķinot $G_1, G_2, ...G_T$ atbilstoši katram solim var koriģēt vērtības tenzorā

$$Q^{i+1}(s_t, a_t) = f(Q^i(s_t, a_t), G_t) \tag{1.27}$$

Kaut gan šai metodei ir teorētiskas konverģences garantijas pēc pietiekama iterāciju skaita, ļoti strauji pieaug tās modeļa — tenzora Q — parametru skaits, pieaugot iespējamo stāvokļu un darbību skaitam — nepieciešams atsevišķi optimizēt katru iespējamo kombināciju, iespējams, ļoti daudzās iterācijās. Tāpēc praksē parasti tiek lietoti modeļi, kas aproksimē $v_\pi(s)$, piemēram, aģenta-kritiķa (*actor-critic*) neironu tīkli, kas reizē iemācās paredzēt gan sagaidāmo vērtību, gan labāko darbību katram stāvoklim ar potenciāli daudz kompaktāku modeli.

Lai uzdevumu varētu risināt, nepieciešams spēt izteikt kādu analītisku funkciju, kas apraksta pašreizējā stāvokļa tīkamību — izšķir labus rezultātus no sliktiem, vai starpstāvokļiem. Robotikā var būt sarežģīti šādu funkciju izdomāt, turklāt tā var būt ļoti "retināta" stāvokļu-darbību telpā, t.i., tikai ļoti nelielam skaitam (vai ar ļoti nelielu varbūtību blīvumu) sasniegtā stāvokļa atdeves funkcija $R(s_t)$ pieņem nenulles vērtību. Tieši šādu trūkumu mēģina risināt metodes, kas kombinē ekspertu demonstrāciju aproksimēšanu ar adaptīvu pielāgošanos[15]

1.3.5. Robotikas uzdevumi

Darbā ar robotiem uzdevums parasti ir vēlamas paša robota un citu vidē atrodamo objektu telpiskās konfigurācijas sasniegšana, vai virkne ar šādām pārejām (manipulācijām). Protams, pilnīgu fizikālas vides pašreizējā stāvokļa aprakstu gūt nav iespējams, tāpēc trajektoriju laikrindas vienmēr īstenībā sastāvēs no novērojumiem, nevis stāvokļiem — $((o_1, a_1), (o_2, a_2), ...)$. Novērojumu formas var būt ļoti dažādas — sākot ar ļoti detalizētiem robota izpildelementu konfigurācijas (lineāro vai lenķisko pārvietojumu, ātrumu, paātrinājumu, slodžu) aprakstiem, beidzot ar video bez nekādas anotācijas.

1.4. Pētniecības virzienu tematisks dalījums

Varētu sacīt, ka tieši par atdarinošo mašīnmāīšanos rakstīts ir samērā maz. Noteikti, ja salīdzina ar vispārīgākām metodēm vai rīkiem. Taču pat "samērā maz" tomēr nozīmē ļoti lielu publikāciju skaitu, kas apraksta pētījumus ļoti dažādos virzienos. Turklāt robotika dominē kā pielietojuma mērķis šādām metodēm. Lai radītu priekštatu par nozares pašrei-zējo stāvokli un aptuvenu vēsturi, nolemts izšķirt trīs aptuvenus virzienus, kas labi apraksta lielu daļu no pētījumiem par iespējām robotus apmācīt ar piemēriem:

1) trajektoriju kopēšana — mērķi šeit pamatā ir panākt robustu, precīzu atdarināšanu ar nelielām treniņa datu kopām, ja pieejama nepieciešamā informācija par sistēmas stāvokli;
2) novērojumu atdarināšana — ne vienmēr ir pieejami dati padevīgā formā, lai tiešā veidā varētu imitēt tajos veiktās darbības. Plaša pētījumu joma nodarbojas tieši ar trajektoriju iegūšanu no video datiem;
3) adaptīvu un atdarinošu metožu kombinācija — atdarinošās mācīšanās pielietojums, lai uzlabotu stimulēto, un otrādi. Kā panākt, ka neaprobežojamies ar tikai piemēros esošo un spējam pielāgoties? Kā efektīvi uzsākt stimulēto mācīšanos ļoti retinātās atdevju telpās?

2. LĪDZŠINĒJIE PĒTĪJUMI

Šīs nodaļas mērķis ir izveidot aptuvenu nozares pētniecības vēsturisku pārskatu; aprakstīt galvenos sasniegtos rezultātus, gūtās atziņas katrā no tematiskajiem apakšvirzieniem. Protams, ne visus pētījumus iespējams vienkārši klasificēt pēc to piederības šeit izvēlētajām kategorijām, un daudzi varbūt tajā vispār neiederas — taču cenšoties gūt personisku izpratni par kādu tēmu, lai motivētu tālākus pētījumus, ir svarīgi nostatīt iepriekšējus rezultātus to kontekstā, saprast, kāpēc tieši šobrīd aktuālie pētniecības virzieni ir tādi, kādus tos varam redzēt kādā akadēmisko publikāciju datubāzē vai neseno pētījumu pārskatā.

Savā ziņā varētu teikt, ka trīs nodaļās nostādītie mērķi kopā būvē pamatus atdarināšanai kā praktiski izmantojamai modeļu apmācības metodei — vai vismaz tādu priekštatu ir ērti sev radīt, lai labāk orientētos savstarpējās atkarības attiecībās starp to sasniegšanai veltīto pētījumu rezultātiem.

2.1. Trajektoriju kopēšana

Pirmā, varētu teikt galvenā taču ne vienmēr vienkāršākā problēma, ir atrast veidu, kā piejamās ekspertu zināšanas — robotikas kontekstā tās parasti būs pareizas trajektorijas dažādu pārvietojumu un smalku manpiulācijas uzdevumu risināšanai — tiešā veidā atdarināt. Šo procesu mēdz saukt arī par programmēšanu ar demonstrācijām (PBD — *programming by demonstration*)[16, 17]. Idealizētā vidē ar determinētām stāvokļu pārejām un pilnīgu informāciju par tās pašreizējo konfigurāciju šis uzdevums varētu būt pat triviāls, taču praksē saskaramies ar problēmām:

1) darbs notiek ar novērojumiem, nevis stāvokļiem. Pat ja pieejami, piemēram, trajektoriju ieraksti, bieži vien trūkst svarīgas informācijas (varētu būt zināma trajektorijas kinemātika, bet ne tās dinamika — paātrinājumi, bet ne spēki);
2) atšķirības vidē: izpildelementos — varbūt robots ir nedaudz citāds; apkārtnē — varbūt manipulējamo objektu masas, forma vai izvietojums ir nedaudz atšķirīgi no demonstrācijās esošajiem;
3) ja trajektoriju ģenerējis eksperts, kam, iespējams bijusi pieejama informācija, kuras aģentam nav — piemēram, manipulāciju veicis cilvēks ar redzi, bet robotam pieejami tikai kontakta sensori.

Problēmas faktiski nozīmē to, ka reālā sistēmā stāvokļu pārejas nav determinētas attiecībā pret novērojumiem un darbībām. Lai labāk saprastu šos trūkumus, vispirms noderīgi ir aplūkot "naivākos" veidus, kā varētu imitēt piemērus.

2.1.1. Vienkāršas metodes

Pirmais, ko varētu darīt, ir tiešā veidā ierakstīt trajektoriju un to atkārtot. Šī nebūt nav jauna ideja — gandrīz visiem mūsdienu industriāliem robotiem ir pieejamas t.s. *lead-through* un *teach-in* programmēšanas metodes, kas ļauj fiziski un ar tālvadības ierīces palīdzību vadīt robota kustību un to ierakstīt pēcākai atdarināšanai[18], turklāt tās parādījušās jau pašos industriālās robotikas pirmsākumos 1970os gados[19].

Darba autors var pats personīgi izdarīt zināmus secinājumus par tiešu trajektoriju ierakstīšanu un atkārtošanu, jo ir strādājis kā mehatronikas inženieris uzņēmumā, kas nodarbojas ar rūpnieciskās ražošanas iekārtu projektēšanu, izgatavošanu un automatizāciju, tāpēc pietiekami daudz nodarbojies arī ar robotu programmēšanu. Tā kā trajektorijai jābūt ierakstītai tieši ar robotu, lai tā būtu atkārtojama bez papildus datizraces uzdevumu risināšanas, ir zināma tendence dominēt viegli realizējamiem bet varbūt ne optimāliem ceļiem telpā — vieglāk ierakstīt dažus pagrieziena punktus un ļaut programmatūrai interpolēt nekā fiziski vadīt robotu visā kustības ceļā.

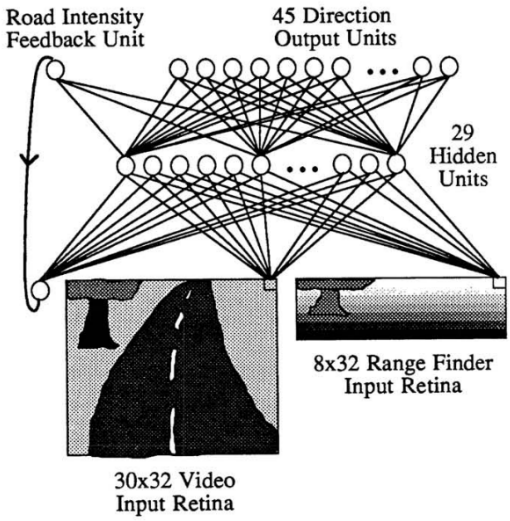
Turklāt var parādīties neparedzēti trūkumi, pārejot no lēnas, nenoslogotas izpildes programmēšanas procesā uz ātru un noslogotu ekspluatācijā, kas apgrūtina procesu. Faktiski sākotnējais ieraksts bieži vien kalpo par starta punktu, bet, lai nonāktu pie lietojamas programmas, nepieciešams iegūto kodu koriģēt un iteratīvi pielāgot. Lai arī principā tiek izmantota demonstrācija trajektorijas iegūšanai, procesa veikšanai tik un tā nepieciešams personāls ar robotu programmēšanas prasmēm. Jau sen atzīts[16, 17], ka, lai tik tiešām robotus varētu apmācīt tikai ar piemēriem, nepieciešamas metodes, kas ir robustākas pret nobīdēm no paraugu ģenerējošā procesa apstākļiem, vispārināmākas, un attiecīgi sākti pētījumi ar mašīnmācīšanās metodēm.

Kad jāspēj atdarināt kas vairāk nekā viena, nemainīga trajektorija, nepieciešams atdarināt nevis pašu trajektoriju, bet gan procesu, kas tādas ģenerē — "eksperta" stratēģiju. Viena no vienkāršākajām metodēm, kas bieži tiek lietots kā piemērs, taču praksē reti kad ir pielietojuma, ir uzvedības klonēšana (*behavioural cloning*). Vispārīgi to definēt ir samērā vienkārši[10]. Ja dots MDP un kāda eksperta stratēģija π^*, kas šo MDP optimāli risina, mērķis ir atrast maksimāli tuvu modeli π_θ, kur

$$\pi_\theta(s) \approx \pi^*(s) \tag{2.1}$$

Parasti, protams, ir pieejama datu kopa ar eksperta izietajām stāvokļu-darbību laikrindām, turklāt jāstrādā ir ar novērojumiem, nevis stāvokļiem. Kā ilustratīvu piemēru mēģinājumam realizēt šādu algoritmu bez īpašām korekcijām var izmantot 1989. gadā Kārnegija-Melona Universitātē veikto pētījumu "*Autonomous Land Vehicle in a Neural Network*" (ALVINN)[20]. Tā mērķis bija izstrādāt pašbraucošu automašīnu, kas spēj sekot ceļa kontūram.

Automašīna tikusi aprīkota ar videokameru un LIDAR sensoriem, kas devuši divus skatus uz to pašu telpas reģionu automobiļa priekšā. Par apmācāmo modeli izvēlēts neironu tīkls. Protams, 1989. gads vēl bija laiks, kad datoru veikstpēja bija stipri ierobežota, un nevienam vēl nebija ienācis prātā būvēt tik dziļas, daudzskaitlīgas un sarežģītas tīklu arhitektūras kā mūsdienu konvolucionālos tīklus vai transformatorus. Tāpēc neironu tīkls ir gaužām līdzīgs jebkurā mācību grāmatā pirmajā nodaļā atrodamajiem piemēriem — tam ir viens slēptais slānis ar 29 perceptroniem, kam seko 45 izvades elementi. Video izmantots krāsainā attēla zilais kanāls, jo tajā ceļa virsma visvairāk kontrastē ar apkārtējo vidi. Gan video, gan LIDAR radītie attēli tīkla ievadē veido vienkāršu vektoru bez nekādiem telpiskiem kodējumiem, visi slāņi savstarpēji pilnībā savienoti.



Att. 1: ALVINN modeļa uzbūve[20]

Modeļa izvades slānis apzīmē vēlamo stūrēšanas virzienu 45 diskrētos soļos. Treniņa datu kopā faktisko virziena komandu atspoguļo neprecizēta veida "zvana" funkcija ar modu pie pareizā virziena. Ieviests viens papildus perceptrons, kas (teorētiski) novērtē ceļa gaišumu salīdzinot ar apkārtējo vidi, un tiek pievienots nākamās iterācijas ievades vektoram.

Jau šim (šķietami) samērā vienkāršajam uzdevumam konstatēts, ka ievākt treniņa datus fizikālā vidē — braucot ar automašīnu pa ceļiem un ierakstot vadītāja veiktās korekcijas — nav praktiski, jo nepieciešama ļoti liela treniņa datu kopa. Jāatzīst, ka ar modernākiem tehniskās redzes modeļiem droši vien šī nepieciešamība mazinātos. Tāpēc dati ģenerēti sintētiski — tā kā gan video, gan attāluma datu izšķirtspēja ir gaužām neliela, pat ar 1989. gadā pieejamām datorgrafikas iespējām šādi gūtus attēlus ir grūti atšķirt no īstiem. Simulatorā iegūtie attēli un vadības komandas izmantoti klasifikatora apmācībā.

Iegūtais rezultāts — modelis, kas maksimāli tuvināts simulatorā realizētajam kontroles algoritmam izmantotā šablona iespēju robežās. Tas bijis pietiekami labs, lai spētu vadīt ar kameru un attāluma sensoru aprīkotu automobili pa 400m garu slēgta ceļa posmu saulainos dienas apstākļos, ar ātrumu 0,5m/s. Tas tiek lietots kā arguments par neironu tīklu pavērtajām iespējām pašbraucošo auto attīstībā, taču netiek slēpts, ka sasniegtais ir tālu no praktiskas vadības sistēmas.

Kā galvenais uzvedības klonēšanas trūkums parasti tiek minēta nespēja atgūties no faktiskā stāvokļa sadalījumu nobīdes[10] (*distribution shift*). Ja reālais modelis $\pi_\theta(s)$ nevar pilnīgi precīzi atdarināt eksperta $\pi^*(s)$ darbības, sākotnējais sistēmas stāvoklis ir atšķirīgs no tiem, kas pārstāvēti treniņa datu kopā vai (iespējams, visbiežāk) stohastiskas MDP pārejas funkcijas gadījumā treniņa datu kopa neietver visas iespējamās trajektorijas ar atbilstošajām $\pi^*(s)$ vērtībām. Lai iegūtu precīzāku un robustāku eksperta stratēģijas atdarinājumu, piedāvāti dažādi — sarežģītāki — apmācības paņēmieni.

3. PRAKTISKA REALIZĀCIJA - RĪKI, PIEMĒRI

3.1. Simulācijas vides un saskarne

3.2. Vienkāršu modeļu realizācijas individuālai izpratnei

3.2.1. Stimulētā mašīnmācīšanās

3.2.2. Uzvedības kopēšana

3.2.3. DAgger

4. IEVADS

4.1. Uzdevums

Datizraces uzdevumi var būt dažādi. Viens no veidiem, kā tie var radikāli atšķirties pēc būtības, ir pirms pētījuma sākšanas pastāvošā skaidrība par rezultāta formu. Ja kādreiz sākam ar datu kopu, par ko nekas nav zināms, un mēģinām gūt vispārīgus priekštatus par starp tās elementiem pastāvošajām sakarībām, citreiz jau no paša sākuma ir laba izpratne par to, ko vēlamies sasniegt, kādi ir datu kopu veidojošie mehānismi un kādus skaitļus varētu redzēt iegūto aprakstošo modeļu parametros. Šis uzdevums visai pārliecinoši pieder otrajai kategorijai. Prasīts atbildēt uz ļoti specifisku jautājumu. Tāpēc tā vietā, lai sāktu ar ļoti vispārīgu datizraces metožu lietojumu, varam pielāgot vai izstrādāt darba rīkus tieši viena jautājuma atbildēšanai. Turklāt jautājums uzdots par datu kopas ģenerējošo procesu - eirovīzijas dziesmu konkursu - nevis par kādu konkrētu, specifiskā veidā strukturētu korpusu, kas ļauj patstāvīgi izvēlēties maksimāli piemērotu informācijas avotu, ierobežot definīcijas apgabalu pēc saviem ieskatiem, u.t.t.

4.1.1. test

Intuitīvi uzreiz rodas priekštats, kas domāts ar terminiem "kaimiņu būšana" un "objektīvāks novērtējums". Taču ar to nepietiek, lai iegūtu kaut kādu šo fenomenu skaitlisku izteiksmi. Nepieciešams definēt "objektīvu novērtējumu" un attiecīgi - novirzes no tāda. Viens veids, kā to darīt, varētu būt ieviest latentu dziesmu "popularitātes" mēru. Tādu var modelēt, iztēlojoties "demokrātisku" visu skatītāju balsošanu par, viņuprāt, labāko konkursa kārtas dalībnieku dziesmu:

$$i \in \{1, 2, ..., K\} = [K] \tag{4.1}$$

$$N_i - \text{balsis par dziesmu}; N - \text{balsis kopā} \tag{4.2}$$

$$q_i = \frac{N_i}{N}; \sum_{i \in [K]} q_i = 1 \tag{4.3}$$

$$s_i \in [10] \cup 0 - \text{punktu skaits (score)}; s_i \sim P(s_i = x \mid q_i, K) \tag{4.4}$$

$$q_i \leq q_j \Rightarrow s_i \leq s_j \tag{4.5}$$

$$s_i, s_j \neq 0 \Rightarrow s_i \neq s_j \tag{4.6}$$

kur $P(x \mid q_i, K)$ ir sadalījums, kas apraksta katra iespējamā diskrētā novērtējuma (punktu skaita) varbūtību, pieņemot, ka dziesmas "demokrātiskā" balsojuma varbūtība ir q_i. Šķiet, ka šis sadalījums varētu kaut kādas formas binomiālais, (vai arī kaut kas krietni sarežģītāks), taču tā precīzā analītiskā forma nav svarīga tālākiem aprēķiniem. Svarīgi piebilst, ka eirovīzijas vērtējumu sistēmā parasti punkti pieder kopai $[8] \cup \{0, 10, 12\}$, un šīm skaitliskām vērtībām ir nozīme, rēķinot gala rezultātu (punkti tiek skaitīti kopā), taču katras dalībvalsts vērtējuma piešķiršanas procesā šiem skaitļiem ir tikai ordināla nozīme, t.i., $10 < 12$; $\forall i \in [8] : i < 10$. Tāpēc var pieņemt, ka $s_i \in [10]$ un vajadzības gadījumā izmantot pārveidojumu $\hat{s}_i = f(s_i); f(9) = 10; f(10) = 12; f(x \neq 9, 10) = x$.

Tad sagaidāmo punktu skaitu, ja balsojums notiek tikai vienreiz un sakrīt ar "objektīvo" novērtējumu (vienā valstī, visās kopā), var izteikt kā:

$$E[s_i^1] = \sum_{s \in [10]} P(s \mid q_i, K) * s \tag{4.7}$$

bet, ja balsojums tiek atkārots vairākas reizes un tiek skaitīta to svērto variantu summa (K' ir balsojošo dalībvalstu skaits, kas daudzreiz ir tāds pats, kā uzstājošos dalībvalstu skaits, bet ne vienmēr, jo nesenākos konkursos ieviesta pusfinālu sistēma un visas valstis var balsot finālā):

$$E[\hat{s}_i^{K'}] = \sum_{j \in [K']} \sum_{s \in [10]} P(s \mid q_i, K) * f(s) \tag{4.8}$$

Ja interesē nevis dziesmas rezultāts konkursa uzvarētāja noteikšanai, bet tās vidējā ordinālā pozīcija katras balsotājvalsts vērtējumā, pārveidojumu $f(x)$ var (varētu pat teikt, ka nepieciešams) atmest. Pilnīgi korekts novērtējums šis nav tāpat, jo pazudušas ir visas vērtības, kas nav starp 10 lielākajām:

$$E[s_i^{K'}] = \sum_{j \in [K']} \sum_{s \in [10]} P(s \mid q_i, K) * s = K' * E[s_i^1] \tag{4.9}$$

Kā redzams, rezultāts nav atkarīgs no katras dalībvalsts un ir "objektīvs". Ieviest nobīdes nacionālajos balsojumos varētu ar svariem:

$$E[s_i^{K'}] = \sum_{j \in [K']} E[s_i^1] * w_{ji}, \sum_{i \in [K]} w_{ji} = 1 \tag{4.10}$$

un tad "neobjektivitāti" varētu potenciāli labot, lai atjaunotu sagaidāmo vērtību, reizinot svarus ar korekcijas koeficientiem, kas iegūti, dalot svarus ar vienmērīgam sadalījumam (pār vērtējamām dalībvalstīm, nevis balsojošajām) atbilstošajiem:

$$c_{ji} = \frac{w_K^0}{w_{ji}} = \frac{\frac{1}{K}}{w_{ji}} = \frac{1}{K * w_{ji}} \tag{4.11}$$

un visu kopā apkopot korekciju matricā:

$$C = \begin{bmatrix} c_{11} & \dots & c_{1K'} \\ \vdots & \ddots & \vdots \\ c_{K1} & \dots & c_{KK'} \end{bmatrix} \tag{4.12}$$

Lai šo korekciju matricu pielietotu rezultātu labošanai, punktu matricu (bez $f(s)$ pārveidojuma) pa elementiem reizina ar korekciju matricu:

$$S = \begin{bmatrix} s_{11} & \dots & s_{1K'} \\ \vdots & \ddots & \vdots \\ K1 & \dots & s_{KK'} \end{bmatrix} \tag{4.13}$$

$$S' = C \circ S \tag{4.14}$$

Kas tad īsti ir iegūts, un kā no tā aprēķināt konkursa rezultātu? Jāatceras, ka s_i ir punktu skaits, kas iegūts, pēc slēptā mainīgā q_i kārtojot konkursa dalībniekus un piešķirot punktus 10 labākajiem. Ir izdarīts pieņēmums, ka katra valsts vispirms ieguvusi šos punktu skaitus no vienādiem varbūtību sadalījumiem, tad pareizinājusi ar svariem. Koriģējot, atgriezta pēdējā darbība, un iegūti punktu skaiti, kādi tie būtu pirms šīs fiktīvās svēršanas operācijas, pēc fiktīvas demokrātiskas balsošans un punktu piešķiršanas.

Protams, ka realitātē process ir citāds: svari netiek pielietoti punktiem, tā vietā jau punktu skaitu sadalījums ir kropļots - precīzākus rezultātus varētu iegūt, rēķinot korekcijas saņemto balsu skaitiem, ja tie būtu zināmi (un nebūtu žūrijas komponentes, kas visu šo "demokrātijas" modeli padara par vienkārši nederīgu). Turklāt visas vērtības, kas nav bijušas augstākajā desmitniekā no datu kopas vienkārši ir izgrieztas (vienādas ar 0). Tāpēc jau uzreiz var pateikt, ka no matemātiska viedokļa, ar šādu matricu nav iespējams atjaunot slēpto q_i sadalījumu. Taču no statistikas kursa pagājušajā semestrī zināms, ka t.s. "rangu" metodes, kas strādā ar kārtas skaitļiem, nevis skaitliskām vērtībām tiešā veidā, parasti uzvedas vismaz virpsusēji līdzīgi nepārtrauktajām, un bieži vien ir algoritmiski vienkāršākas (pat ja kaut ko par tām pierādīt mēdz būt grūtāk), tāpēc var pastāvēt zināma cerība, ka pat matemātiski nekorektas un nepilnīgas, no datu korpusiem ar iztrūkumiem iegūtas korekcijas varētu darboties vismaz pareizajā virzienā.

Attiecīgi tiek izvirzīts sekojošs korekcijas modelis: tā kā iegūtas ir "atjaunotās" rangu sagaidāmās vērtības, tās drīkst vienkārši pārkārtot - piešķirt kārtas skaitļus no mazākās uz lielāko - un izdarīt korekciju $f(s)$, lai svērtu kārtas skaitļus summās starp balsojošajām valstīm.

$$S'' = S' \text{ rangos 1-10 (pārējie = 0), pa kolonnām} \tag{4.15}$$

$$\hat{S}'' = f(S'') \tag{4.16}$$

$$\hat{s}_i = \sum_{j \in [K']} \hat{s}_{ij} \tag{4.17}$$

Jāmin, ka šādi nav iespējams izšķirt tieši "kaimiņu būšanu" - ko varbūt gribētos rakstu-rot kā tīri etniskas tuvības vai ģeopolitisku interešu sakritības motivētu nobīdi

balosjumu rezultātos. No citiem faktoriem, kas arī atšķiras valstu starpā - kulturālas noslieces, demogrāfiskie sadalījumi, konkursa popularitāte, u.t.t. - radušās nobīdes skaitliski izskatītos tāpat.

4.2. Datu kopas

Kā jau minēts iepriekš, uzdevums ir par fenomenu, nevis tā radītu konkrētu datu kopu. Tāpēc iespējams ne tikai brīvi pēc saviem ieskatiem pārveidot vienu datu kopu, bet apzināti meklēt un izvēlēties jau maksimāli atbilstoši noformētu. Nav arī dots stingrs uzstādījums, ka obligāti jāstrādā ar visu konkursa vēsturi. Laika gaitā ir notikušas daudzas noteikumu un organizatoriskas izmaiņas, kas var apgrūtināt dažādu periodu rezultātu salīdzināšanu.

Par datu kopu izvēlēta *Eurovision Song Contest Dataset* (pieejama *GitHub* repozitorijā), kur jau atrodams korpuss *votes.csv*. Tajā katrā rindā dots notikuma gads, atbilstošā stadija (fināls; pusfināls; pirmais vai otrais pusfināls gados, kad ir divi), vērtējošā valsts un punktus saņemošā valsts.

Pietiek vien atvērt *Wikipedia* rakstu par konkursa balsošanas kārtību, lai kristu nelielā panikā. Garākais (un, Latvijas iedzīvotājiem, interesantākais) posms ar samērā noturīgu balsošanas kārtību ir 1980-2015, tāpēc tālāk tieši ar to arī pārsvarā strādāts.

5. METODES

5.1. Modeļi

Sadaļā 1.1. aprakstīts, kā varētu izskatīties korekcijas matrica, un radīta aptuvena nojausma par procesu, kas šādu matricu varētu ģenerēt, taču tā nav konstruktīva - svari w_{ji} *a priori* nav zināmi, tos nepieciešams noteikt empīriski. Līdz šim arī aplūkots tikai gadījums ar vienu q_i "kvalitātes vērtību" sadalījumu. Dažādās konkursa kārtās šie sadalījumi var radikāli atšķirties. Vienam konkursa etapam korekcijas varētu vienkārši algebriski izrēķināt, taču, ja vēlamies noteikt korekcijas matricu C noteikt globāli, jāveido modeļa šablons un jāapmāca.

Var krietni palauzīt galvu, domājot par veidiem, kā to izdarīt. Pirmā ideja, kas varētu rasties, varētu būt vienkārši ņemt punktus tiešā veidā no datu kopas un apmācīt klasifikatoru formā (vērtējošā valsts, vērtējamā valsts) $\rightarrow$ vērtējums (kur vērtējums vai teksta mainīgai, vai diskrētu vērtību vektors, kur 1 apzīmē konkrētu punktu skaitu). Lai iegūtu sagaidāmās vērtības novērtējumu, var izmantot daudzu klasifikatoru īpatnību - modeļa ģenerētais izejas vektors var tikt normalizēts uz varbūtību sadalījumu, nevis vienu konkrētu klasi. Šīs varbūtības tad var reizināt ar atbilstošajām punktu vērtībām, lai iegūtu matemātisko cerību. Problēma ar šādu pieeju ir tāda, ka bieži vien pieejami pavisam nedaudzi novērtējumi no vienas valsts uz otru. Par spīti šim trūkumiem, mēģinājumi uztrenēt klasifikatora šablonu datu kopai tika veikti, bet vienīgais, kas sniedza cilvēkam saprotamus rezultātus, bija neironu tīkls. Iegūtās sagaidāmās vērtības var tikt izmantotas kā distances starp dalībvalstīm vai par estimatoru, no kā tālāk rēķināt korekcijas. Citi izmēģinātie modeļu šabloni bija SVM un *NaiveBayes*, taču ar tiem semantiski saprotamas skaitliskas vērtības gūt neizdevās, un tiem atbilstošais kods ir izkomentēts.

Tā kā neviens no tipveida modeļu šabloniem nešķita ideāli piemērots tieši šim uzdevumam, pirmais algoritms, kas tika realizēts, bija paša rakstīts. Tā vietā, lai pakāpeniski optimizētu šablonu, iterējot pār datu kopas elementiem, datu kopu var mēģināt reducēt uz formu, kur rezultāts atrodams algebriski. Šajā konkrētajā gadījumā tas darīts, nosakot vidējās empīriskās vērtības piešķirto punktu skaitam katram valstu pārim katrā virzienā, pārveidojot tās par divdimensionālu varbūtību sadalījumu un pielīdzinot rezultātu vienmērīgajam sadalījumam.

To dara, atrodot divus korekcijas koeficientus: r_i, kas vienādo rindu varbūtību summas (savā ziņā kompensējot q_i) un c_{ij}, kas vienādo varbūtības koriģētās matricas kolonnās un reizē ir arī korekcijas matricas vērtības.

$$p_i = \sum_{j \in [K']} p_{ij} \tag{5.1}$$

$$p_i * r_i = \frac{1}{K} \tag{5.2}$$

$$r_i = \frac{1}{p_i * K} \tag{5.3}$$

$$p'_{ij} = p_{ij} * r_i \tag{5.4}$$

$$p'_{ij} * c_{ij} = \frac{\sum_{i \in [K]} p'_{ij}}{K} \tag{5.5}$$

$$c_{ij} = \frac{\sum_{i \in [K]} p'_{ij}}{p'_{ij} K} \tag{5.6}$$

$$w_{ij} = \frac{1}{c_{ij} K} \tag{5.7}$$

Praktiski tika konstatēts, ka, iespējams, kvalitatīvākus rezultātus sniedz korekcijas kvadrātsakne, par ko diskutēts pie rezultātiem:

$$c'_{ij} = \sqrt{c_{ij}} \tag{5.8}$$

$$S' = C' \circ S \tag{5.9}$$

Par šī algoritma matemātisko pareizību pārliecības nav nekādas - pirmais kompensāci-jas solis koriģē q_i, balstoties uz stipri kropļotiem vērtējumiem, un svari tad tiek rēķināti no šī nekorektā starprezultāta. Taču vismaz virspusēji šķiet, ka iegūie rezultāti varētu būt noderīgi.

Iespējams, ka varētu šo procesu pilnveidot, ieviešot iterāciju un cerot uz konverģenci, atkārtoti veicot korekcijas aprēķinu koriģētiem datiem, taču matemātiski pamatot, kāpēc tam būtu jāstrādā, laika nav pieticis. Galvenais šķērslis ir šaubas par to, kā pareizi kombinēt soļos iegūtās korekcijas matricas - vai tas vispār ir pamatojams. Taču zināms, ka līdzīgas metodes izmanto citur datizracē, lai cīnītos ar t.s. *distribution shift* jeb atšķirību starp modeļa ģenerētiem sadalījumiem un reāliem.

5.2. Datu priekšapstrāde

Klasifikatoru šablonu izmantošanai nepieciešams datu kopu "vektorizēt", t.i., diskrētas klases ieejas datu kopā izteikt kā vektorus ar vienu nenulles elementu. Atkarībā no konkrētā algoritma realizācijas, var būt nepieciešams rezultātu kolonnu vai nu izteikt kā tādu pašu vektoru, vai viendimensionālu sarakstu ar teksta elementiem. Šīs manipulācijas veic *Python* skripts *vectorize.py*, pieejams kopā ar visiem pirmkoda failiem un dažādiem datu apstrādes starpproduktiem projekta repozitorijā.

Lai veiktu aprēķinus pēc algebriskās metodes, jāveic smagnējāki pārveidojumi. Kopu ar kortežiem formā (t, i, j, s) nepieciešams izteikt kā tensoru ar elementiem s_{ijt}, kur t - konkursa etaps. Ja konkursa etapu dalībnieki un balsotāji vienmēr būtu tie paši, šis būtu triviāls uzdevums. Taču sarežģījumus rada fakts, ka neeksistējošas vērtības un nulles vērtības nav viens un tas pats. Ne katra dalībvalsts piedalās katrā etapā un ne katra dalībvalsts, kas balso, arī piedalās konkursā. Tāpēc neeksistējošas vērtības nepieciešams marķēt atsevišķi. Izvēlētais risinājums ir tās marķēt ar -1, un pēc tam datu apstrādē īpaši apstrādāt šādi marķētus ierakstus, kur nepieciešams. Vērtībām 10,12 tiek ieviesta jau iepriekš aprakstītā korekcija uz 9,10.

Šīs operācijas veiktas repozitorijā pieejamajā skriptā *datagen.py*. Principā priekšapstrādes solī iespējams arī veikt vidējo vērtību rēķināšanu, kas droši vien arī būtu ātrāk

nekā faktiski realizētajā tensora ģenerēšanas pieejā, taču tīri praktiski apsvērumu vadībā tika izlemts visas netriviālās datu manipulācijas atstāt atsevišķi: darba autors daudz labāk pazīstams ar n-dimensionālu homogēnu datu bloku skaitļošanas un lineārās algebras bibliotēku *numpy* nekā ar 2-dimensionālu heterogēnu datu korpusu apstrādes bibliotēku *pandas*. Izstrādes procesā vieglāk iteratīvi izmainīt visu procesu, strādājot ar jau gatavu datu tensoru. Lai nebūtu katru reizi jāveic tensora veidošana, kas aizņem gandrīz 40 sekundes, starprezultāts tiek saglabāts failā. Datu pārveidošanu ir iespējams paātrināt daudzkārtīgi, taču tad jāizmanto cita darbību secība, ko grūtāk atkļūdot. Reizi dažās stundās zaudēt 40s šķiet mazāks zaudējums, nekā pavadīt vairākas stundas, pārrakstot jau pietiekami labi strādājošu kodu.

Šī ir arī laba vieta tekstā, kur norādīt, ka šādam datu formātam matrica ar vērtējošajām un vērtejamām valstīm vienmēr ir kvadrātiska, t.i., $K = K'$. Iztrūkstošās šķautnes ir atbilstoši marķētas ar vērtībām ārpus parastā definīcijas apgabala..

5.3. Modeļu aprēķins

Universālo klasifikatoru apmācībai izmantota bibliotēka *scikit-learn*, kas ļauj ar ļoti lako-niskām definīcijām pielietot dažādus visai jaudīgus datizraces algoritmus. Ja nav vēlmes vai nepieciešamības īpaši iedziļināties modeļu hiperparametru optimizācijā, šādu rīku izmantot ir vienkāršāk nekā daudz jaudīgākas bibliotēkas kā *tensorflow*. Kods atrodams skriptā *model-classifier.py*. Tā kā šī programma rakstīta vēlāk, tā vietām izmanto funkcijas no zemāk aprakstītās.

Algebriskā modeļa aprēķins tiek veikts citā *Python* skriptā - *model.py*. Pirms iespējams aprēķināt vidējās vērtības pār etapiem, nepieciešams noteikt katra balsojuma virziena (varētu teikt, orientēta grafa šķautnes) kopskaitu. To veic funkcija *coincidence_count*, kas atgriež matricu ar vērtībām formā n_{ij} - reižu skaits, kad j-tā valsts balsojusi konkursā, kurā uzstājas i-tā. Šo parametru izmanto funkcija *clear_dataset*, kas atsijā valstis pēc sliekšņa kritērija - ja maksimālais n_{ij} ir mazāks par slieksni, no matricas tiek izmestas i-tās rindas un i-tās kolonnas. Tādējādi tiek apkarota traucējoša parādība - valstīm, kas piedalījušās konkursā tikai dažas reizes, korekcijas koeficienti var būt krietni lielāki vai mazāki, nekā visām pārējām, jo lielo skaitļu likumam nav pieticis elementu, lai izlīdzinātu iegūto punktu skaitus un tuvotos sagaidāmajai vērtībai. Piešķirto punktu summu atrod funkcija *coincidence_total*. Vidēji j-tās valsts i-tajai piešķirto novērtējumu tad atrod *edge_average*.

Nākamais solis ir iegūto vidējo vērtību pārveidošana par varbūtību sadalījumu. Sāku-mā tas netika darīts, un rezultāts bija grūtības ar izsekošanu vērtību semantiskajai nozīmei un pareizu algoritma realizāciju. Strādājot ar varbūtībām, daudzas potenciālas programmatiskas kļūdas var novērst, vienkārši sekojot līdzi tam, ka rindu, kolonnu vai kopējā varbūtību summa matricā ir vienāda ar 1 (atkarībā no veicamajām darbībām). Papildus tiek veikts arī matemātiski pilnīgi nepamatots taču praktiski pašsaprotams solis - visām vidējo vērtējumu matricas vērtībām tiek pieskaitīta konstante α, kas ir funkcijas parametrs. Tas tiek darīts, jo citādi nogrieztām vērtībām tiek aprēķināts svars 0 un

korekcijas koeficients $+\infty$, kas korekcijas mērķiem neder. To var uztvert kā zaudētās informācijas ekstrapolāciju - valstis, kas ne reizi nav saņēmušas rangu 1-10, tik un tā gandrīz noteikti kādas balsis saņemtu, ja tiktu veikta izlase no balsis veidojošā sadalījuma. Visu augstāk minēto veic funkcija *normalized_score*.

Tālāk, 2.1. sadaļā aprakstīto korekcijas matricas aprēķinu veic funkcija *corrections*. Strādājot ar *numpy*, mēdz rasties nepieciešamība veikt skalāras operācijas starp vektoriem, matricām un dažādu dimensiju tensoriem - un tad var pieļaut grūti atrodamas kļūdas, nepareizā secībā savietojot to dažādās dimensijas. Tāpēc funkcija *expand* gluži vienkārši vienreiz atrisina šo problēmu, lai vēlāk programmētājam par to nebūtu jādomā. *weights* un *distance_measure* atgriež attiecīgi noteikto svaru un simetrisko distanču matricas informatīviem mērķiem. Šīs pašas funkcijas izmanto, lai apstrādātu neironu tīkla ģenerētās matemātiskās cerības.

Lai korekciju piemērotu datu kopai, izveidotas funkcijas *apply_correction_matrix* un *rank_corrected*. Pirmā pareizina datu kopu ar korekcijām, saglabājot iztrūkstošo datu vērtības, otrā veic diezgan piņķerīgo koriģētu vērtību pārkārtošanas procedūru, ko nevar pilnībā vektorizēt (t.i, uzreiz nodot aprēķina formulu visam tensoram).

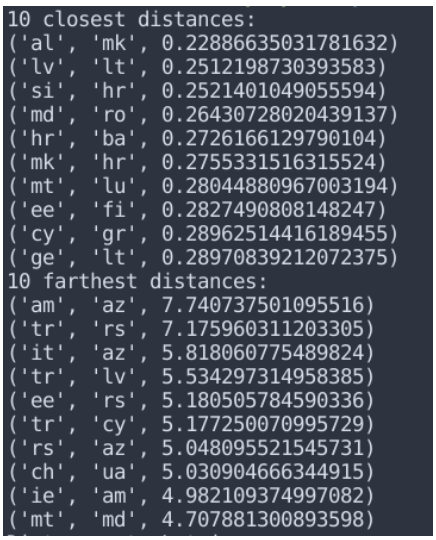
Subjektīvam novērtējumam ar tiktāl minēto pietiek - var aprēķināt matricas koeficientus, pārveidot tos par distancēm, pārbaudīt iegūtās vērtības skaitliski vai veikt ar tām dimensiju redukciju. Taču kā spriest par korekciju "pareizību"? Cik lielā mērā tās patiešām kaut ko uzlabo? Tas, protams, ir filozofisks jautājums, un droši vien pietiekami sarežģīts arī no matemātikas viedokļa. Īpaši neiedziļinoties varētu izmantot intuīciju un pieņemt, ka, ja kaut kādā veidā kompensēsim slēptus svarus, mēģinot vienādot ar tiem iegūtos rezultātus, tad šo te iegūto rezultātu dispersijai vajadzētu mazināties. Funkcijas *split_dataset* un *pre_post_variance* realizē "slinko krosvalidāciju". Proti, viena sadala datu kopu nejauši izvēlētās apakškopās, otra aprēķina dispersiju katram tensora "slānim" un atgriež vidējās vērtības - pirms un pēc korekcijas. Lai gūtu krosvalidācijai līdzīgu efektu, šo procedūru var atkārtot daudzas reizes. Padodot par argumentu visu treniņa kopu abās ailēs, iespējams novērtēt dispersijas izmaiņas visai kopai.

Visbeidzot, iegūtos rezultātus izvada failos - distanču, svaru, korekciju matricas; valstu indeksiem atbilstošos kodus; datu tensora formu (lai varētu pareizi to atjaunot citur). Uz konsoles tiek izvadīti "krosvalidācijas" rezultāti visiem datiem, 10 neujaušiem dalījumiem, 10 tuvākās distances, 10 tālākās un Latvijai - 5 tuvākās, 5 tālākās. Repozitorijā atrodas arī skripts *mds.R*, kur bez īpašām pārmaiņām pārkopēts kods no 2.1. mājas darba, lai pārbaudītu, vai ar MDS un isoMDS var iegūt vizuāli uzskatāmus rezultātus.

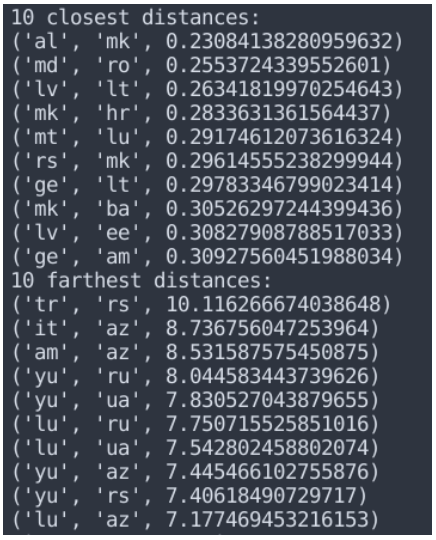
6. REZULTĀTI

Tā kā neironu tīkls apmācīts uz atšķirīgas formas datu kopas, ar to nav mēģināts rēķināt "krosvalidāciju". Bet analoģiski salīdiznāt iespējams abu modeļu radītos distances mērus.

Redzams, ka tuvības starp Balkānu un Baltijas valstīm abi modeļi atrod ļoti līdzīgas (skaitlisko vērtību sakritība gan ir nejauša, nozīme ir secībai - mainot parametru α var

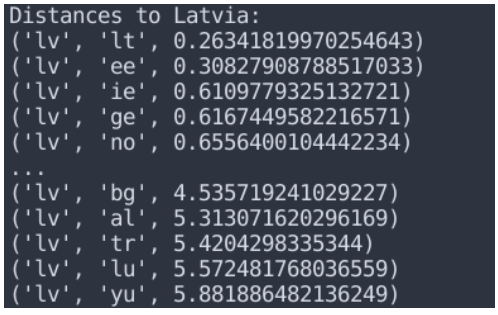


Att. 2: Klasifikatora distances



Att. 3: Vidējo vērtību matricas distances

iegūt dažādas absolūtās vērtības), taču atšķiras valstis, ko modeļi iedala "tālajā galā".



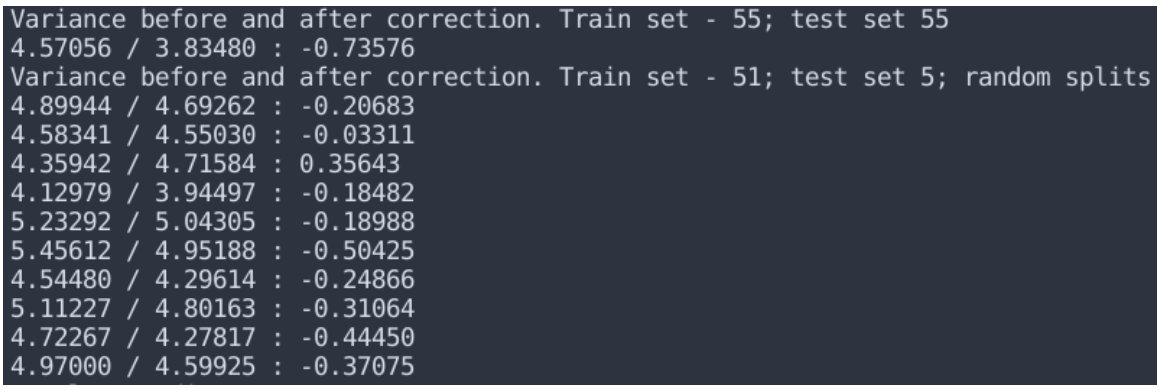
Att. 4: Vidējo vērtību matricas distances - Latvijai

Droši vien pārsteidzošākais secinājums, ja var ticēt iegūtajiem rezultātiem, ir tas, ka pie gandrīz jebkuriem parametriem, viena no stiprākajām "kaimiņu būšanām" ir tieši Latvijai un Lietuvai. Protams, Balkāni veido cieši saistītu grafu, taču savstarpējā fa-

vorītisma ziņā laikam tomēr iepaliek. Tāpat saraksta augšgalā gandrīz vienmēr ir Rumānija ar Moldovu un Albānija ar (Ziemeļ)Maķedoniju.

```
Variance before and after correction. Train set - 55; test set 55
4.57056 / 4.22202 : -0.34854
Variance before and after correction. Train set - 51; test set 5; random splits
4.89944 / 8.90881 : 4.00936
4.58341 / 8.06198 : 3.47857
4.35942 / 9.53019 : 5.17077
4.12979 / 5.81298 : 1.68319
5.23292 / 8.25463 : 3.02171
5.45612 / 7.39811 : 1.94199
4.54480 / 7.22218 : 2.67738
5.11227 / 7.24402 : 2.13175
4.72267 / 7.67969 : 2.95702
4.97000 / 7.53223 : 2.56224
```

Att. 5: Dispersijas izmaiņas - reizinot ar c_{ij}

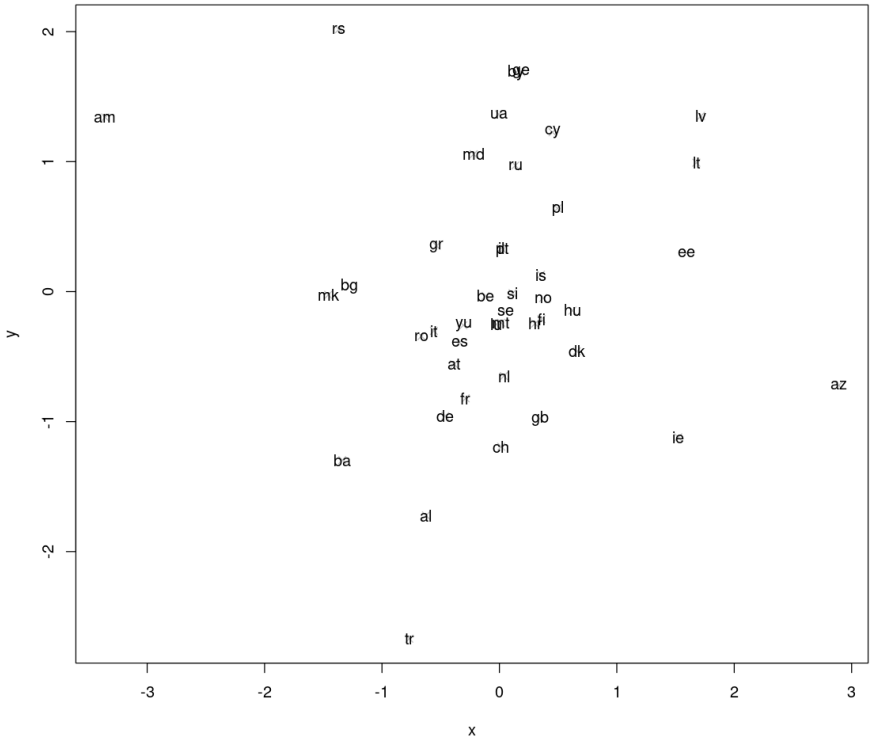


Att. 6: Dispersijas izmaiņas - reizinot ar $\sqrt{c_{ij}}$

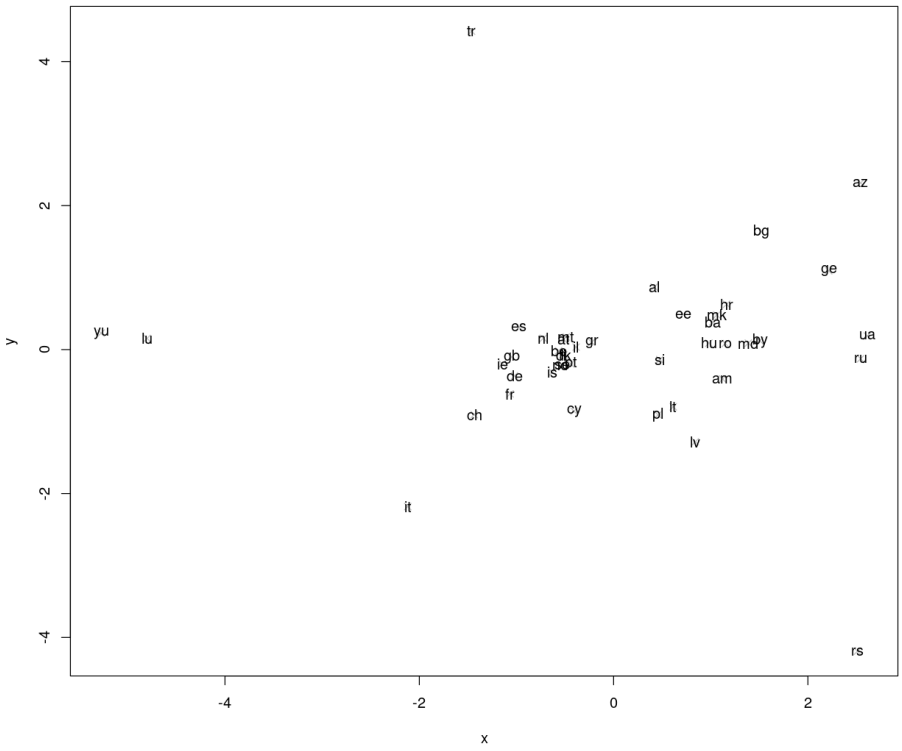
Salīdzinot datu kopu dispersijas (vidējās vērtība korpusam kopumā no 2-dimensionālo sadalījumu dispersijām katra etapa ietvaros), redzams, ka, vienkārši reizinot ar korekcijām, pārkārtojot pašu treniņa kopu un pārrēķinot rangus, dispersija samazinās, taču to pašu darot ar datiem, kas neietilpst treniņa sadalī-jumā, dispersija pieaug. Savukārt reizinot ar korekcijas kvadrātsakni, dispersija krītas abos gadījumos - arī datiem, kas treniņa kopā nav iekļauti. Pastāv divas visai ticamas iespējas:

- Ir kāda vienkārša (vai ne tik vienkārša) matemātiska sakarība, kas determinēti nosaka, ka tā jābūt, bet patstāvīgā darba autors savos neveiklajos un kļūdpilnajos aprēķinos to vienkārši nav pamanījis. Tam varētu būt saistība ar divkāršo koeficientu rēķināšanu - vispirms vienādojot rindu varbūtību summas, tad līmeņojot tās lokāli;
- Notiek kaut kas analoģisks *overfitting* - koriģējot ar pilnajām vērtībām tiek izdarīts par daudz, rezultāti tiek samudžināti un kropļoti. Tieši šī intuīcija ir tas, kas iedvesmoja veikt šādu (autoraprāt) matemātiski nepamatotu pārveidojumu, turklāt pat pirms dispersiju salīdzināšanas. Laimīgas sagadīšanās rezultātā šī korekcija patiešām izrādījās noderīga. Tas ir, ja šāds dispersiju izmaiņas novērtējums vispār kaut ko nozīmē - kas nebūt nav garantēts.

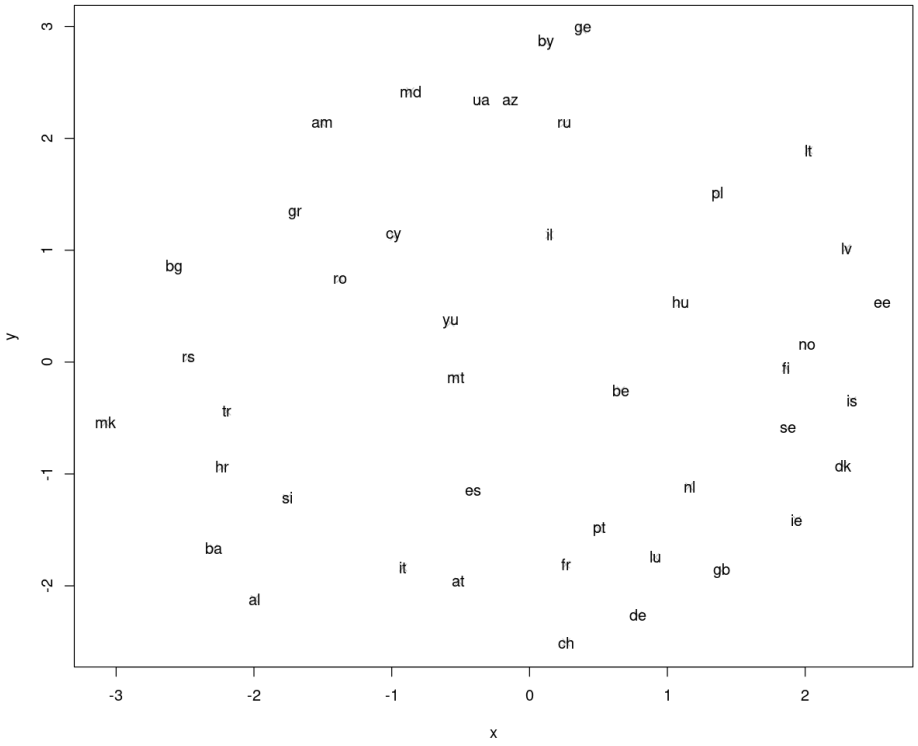
Izmēģināti trīs dažādi vizualizācijas paņēmieni - *MDS*, *isoMDS* un *t-SNE*. Trešais nekādus saprotamus rezultātus sniegt nespēja, tāpēc netiek iekļauts. Pilna izmēra attēli atrodami projekta repozitorijā "tex/" direktorijā, kur ir arī šīs atskaites LaTeXpirmokds.



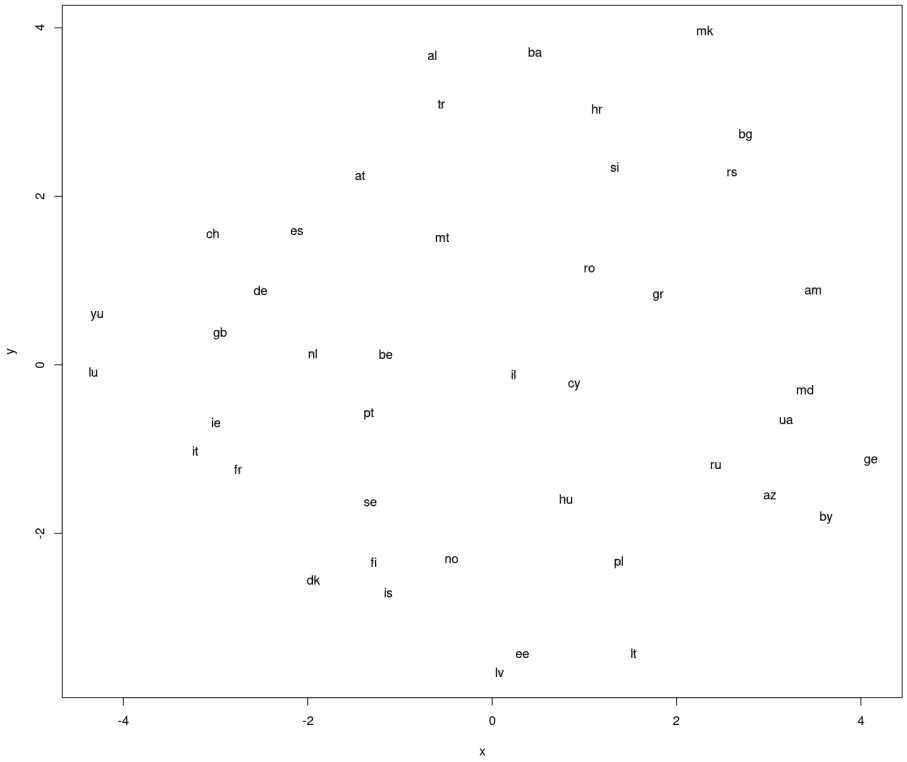
Att. 7: MDS vizualizācija klasifikatora distancēm



Att. 8: MDS vizualizācija klasifikatora distancēm



Att. 9: isoMDS vizualizācija klasifikatora distancēm



Att. 10: isoMDS vizualizācija klasifikatora distancēm

Redzams, ka MDS gadījumā atrastā projekcija ir visnotaļ nevienmērīga, ar klasteriem tuvu koordinātu sākumpunktam un izlēcējiem. *isoMDS* iegūst daudz vienmērīgāku izkliedi. Abos gadījumos tomēr ir redzams, kā, piemēram, Balkānu, Skandināvijas, Austrumeiropas (tās šaurajā definīcijā) vai Baltijas valstis grupējas.

SECINĀJUMI

Darba gaitā tika diezgan detalizēti iepazīta konkrēta datu kopa - Eirovīzijas dziesmu konkursa rezultāti (precīzāk, laika periodā starp 1980. un 2015. gadu, kad vērtēšana notika pēc vienāda principa vai vismaz pietiekami nemainīga principa), izvirzīta hipotēze par ģenerējošo modeli t.s. "kaimiņu būšanas" fenomenam, piedāvāta metode tā koriģēšanai. Praktiski tika izstrādāti skripti gan tipveida klasifikatora ģeneratora pielietojumam, gan datu kopas elementārai algebriskai reducēšanai uz formu, kurā aprēķins būtu izsakāms analītiski.

Rezultātā tika iegūti "kaimiņu būšanas" novērtējumi no diviem radikāli atšķirīgiem modeļiem, kas tomēr daudzējādā ziņā ir līdzīgi. To ģenerēto skaitlisko vērtību un vizualizāciju pārbaude atklāj likumsakarības, kas sakrīt ar cilvēka intuitīvo izpratni par meklējamās parādības būtību. Piedāvātajai korekcijas metodei tika arī izstrādāts kvantitatīvs novērtējums, taču par tā nozīmīgumu ir grūti spriest, jo, tāpat kā pati aprēķinu secība, kas noved pie modeļa, tas nav nekā dziļi matemātiski pamatots.

ATSAUCES

[1] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[2] Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. In: *Master’s Thesis (in Finnish), Univ. Helsinki* (1970), pp. 6–7.

[3] Kunihiko Fukushima. “Neocognitron: A hierarchical neural network capable of visual pattern recognition”. In: *Neural networks* 1.2 (1988), pp. 119–130.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[5] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[6] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[7] Isaac Asimov. *I, robot*. Vol. 1. Spectra, 2004.

[8] J. Grundspeņķis. *Nacionālā enciklopēdija - mākslīgais intelekts*. 2021. URL: `https://enciklopedija.lv/skirklis/24447-m%C4%81ksl%C4%ABgais-intelekts` (visited on 01/14/2022).

[9] Beijing Academy of Artificial Intelligence. *Suggested Notation for Machine Learning*. 2020. URL: `http://ctan.math.utah.edu/ctan/tex-archive/macros/latex/contrib/mlmath/mlmath.pdf` (visited on 01/14/2022).

[10] Alexandre Attia and Sharone Dayan. “Global overview of imitation learning”. In: *arXiv preprint arXiv:1801.06503* (2018).

[11] Abhishek Gupta et al. “Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning”. In: *arXiv preprint arXiv:1910.11956* (2019).

[12] Daniel Brown et al. “Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations”. In: *International conference on machine learning*. PMLR. 2019, pp. 783–792.

[13] Peter Englert and Marc Toussaint. “Learning manipulation skills from a single demonstration”. In: *The International Journal of Robotics Research* 37.1 (2018), pp. 137–154.

[14] Richard S Sutton and Andrew G Barto. “Reinforcement learning: An introduction”. In: MIT press, 2018, pp. 60–77.

[15] Ashvin Nair et al. “Overcoming exploration in reinforcement learning with demonstrations”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6292–6299.

[16] S Muench, J Kreuziger, and M Kaiser. “Robot programming by demonstration (rpd)-using machine learning and user interaction methods for the development of easy and comfortable robot programming systems”. In:

[17] Aude Billard et al. “Handbook of robotics chapter 59: Robot programming by demonstration”. In: *Handbook of Robotics. Springer* (2008).

[18] Alex Owen-Hill. *The Decade of Artificial Intelligence*. 2021. URL: `https://blog.robotiq.com/what-are-the-different-programming-methods-for-robots` (visited on 01/16/2022).

[19] ABB Group et al. “Special report: Robotics–ABB group”. In: *ABB Review* (2016).

[20] Dean A Pomerleau. *Alvinn: An autonomous land vehicle in a neural network*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE and PSYCHOLOGY . . ., 1989.