## Turinys

# Įvadas

Paskutiniais dešimtmečiais sprendžiant prognozavimo uždavinius labai paplito neuroniniai tinklai. Jų pagalba gaunami gana neblogi rezultatai, apibūdinantys tiriamojo požymio priklausomybę nuo pasirinktų faktorių. Neuroninių tinklų universalumas leidžia juos taikyti įvairiose srityse.

Taikant neuroninius tinklus svarbu pasirinkti jų architektūrą, t.y. paslėptų sluoksnių skaičių , neuronų skaičių kiekviename sluoksnyje, aktyvacijų funkcijų tipą.

Yra daug įvairių modelių, besiskiriančių savo architektūra. Darbe aprašomas modelis, naudojantis akyvacijos funkcijas su Gauso branduoliais. Nuo egzistuojančių modelių jis skiriasi tuo, kad kiekvienoje aktyvacijos funkcijoje naudojamos skirtingos diagonalinės kovariacijų matricos ir skirtingi vidurkių vektoriai.

Vertinant nežinomus parametrus naudojamas naujas „boosting search“ algoritmas.
Šio darbo tikslai yra tokie:

1) išbandyti ar gerai klasifikuoja naujas modelis, lyginant su turimais analogiško tipo modeliais (pvz. logistinė regresija).
2) palyginti gautus rezultatus naudojant Gauso aktyvacijos funkcijas ir Renyi enropija maksimizuojancias funkcijas.

Teoriniai rezultatai iliustruojami pavyzdžiais. Vienu atveju bus nagrinėjamos aktyvacijos funkcijos, maksimizuojančios Shannon entropija, kitu – maksimizuojančios Renyi entropiją. Manoma, kad tokio tipo funkcijos turėtų geriau tikti prognozavimui. Parinktų modelių tinkamumas tikrinamas klasifikuojant brokuotus ir gerus kineskopus. Darbe naudojama gamyklos “Ekranas” AB duomenys.

# 1. Neuroniniai tinklai

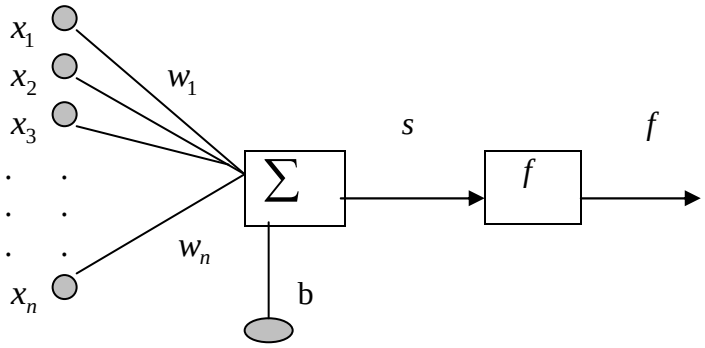
## 1.1 Vienasluoksnis perceptronas (VSP)

Tegu turime priklausomą kintamąjį y, kurio reikšmes norime prognozuoti pagal keletą nepriklausomų kintamųjų $x_i = (x_1, x_2, ..., x_n)$. Tarkime, kad duomenys susieti tokia priklausomybę:

$$y = f(s) + \varepsilon, \quad \text{kur } \varepsilon \text{ -a.d}$$

Toks modelis vadinamas VSP: $s = \sum_{i=1}^{n} w_i x_i + b$, i=1,2,…,n

$w_i$ vadinami svoriais, b-slenksčiu, s-sumavimo rezultatu, y-neorono išėjimo signalas, f-aktyvacijos funkcija.
Paprastasis perceptronas pavaizduotas pav.1




pav.1 Vienasluoksnis perceptronas

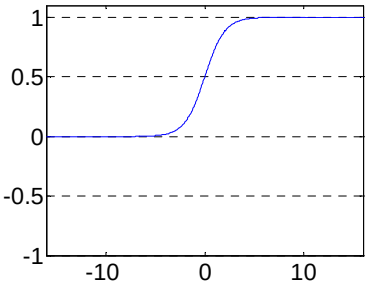
**<u>Aktyvacijos funkcijos:</u>**

Neurono aktyvacijos funkcijos gali būti įvairios, konkrečios aktyvacijos funkcijos parinkimas priklauso nuo konkretaus sprendžiamo uždavinio. Klasikiniu atveju perdavimo funkcija yra sigmoidinė (a). Taip pat dažnai naudojamos: Gauso funkcija (b), hiperbolinis tangentas (c), tiesinė (d) :

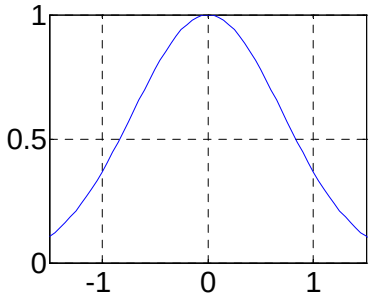
$$f(s)=\frac{1}{1+\exp(-as)} \quad \text{(a)} \qquad f(s)=\exp(-s^2) \quad \text{(b)}$$

$$f(s)=\frac{\exp(as)-\exp(-as)}{\exp(as)+\exp(as)} \quad \text{(c)} \qquad f(s)=\begin{cases} -1, & s\le -1 \\ s, & -1<s<1 \\ 1, & s\ge 1 \end{cases} \quad \text{(d)}$$

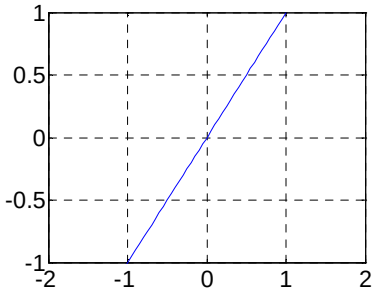
Dažnai aktyvacijos funkcija gali įgyti reikšmes tik iš tam tikro intervalo, kai įėjimų reikšmės kinta nuo $-\infty$ iki $\infty$. Pvz., logistines funkcijos išėjimų reikšmės priklauso intervalui (0,1), hiperbolinio tangento (-1,1), o tiesinės funkcijos *f(s) =ks* reikšmės gali kisti intervale $(-\infty,\infty)$.



Sigmoidinė



Gauso



Tiesinė

Kiekvienas neuronas apibrėžia hiperplokštumą (dvimačiu atveju - tiesę), kuri erdvę padalina į dvi dalis. Hiperplokštumos lygtis:

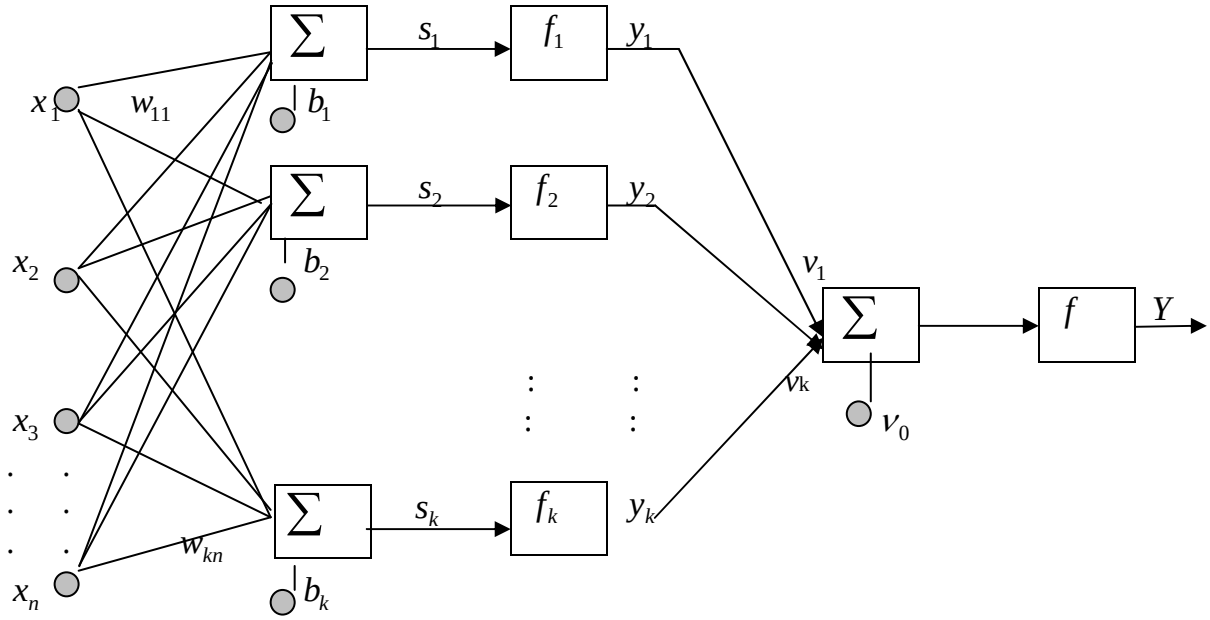
$$\sum_{i=1}^{n} w_i x_i + b = 0$$

Vienasluoksniai neuroniniai tinklai leidžia spręsti tik paprasčiausius klasifikavimo uždavinius, nes skiriamiji paviršiai yra labai paprasti. Modifikuoti perceptronai, t.y. daugiasluoksniai neoroniniai tinklai kaip taisyklė turi daugiau galimybių, yra naudojami sprendžiant sudėtingus uždavinius, nes duoda galimybę sudaryti sudėtingus skiriamuosius paviršius .

## 1.2 Daugiasluoksnis perceptronas (DSP)

Daugiasluoksnis perceptronas (DSP) gaunamas jungiant VSP į sluoksnius. Šią NT architektūrą 1986m. pasiūlė Rumelhart ir McClelland.

Nagrinėsime dvisluoknį perceptroną (pav.3).




pav.3 Daugiasluoksnis peceptronas

Tarkim DSP turi *n* įėjimų $\mathbf{x} = (x_1, x_2, ..., x_n)^T$. Tada neurono paslėptame sluoksnyje duodamas išėjimas $y_j$, $j$=1,2,..$k$ gali būti apskaičiuotas tokiu būdu:

$$y_j = f(W_j^T \mathbf{x} + w_{j0}),$$

čia $j$ - neurono numeris paslėptame sluoksnyje; $k$ - neuronų skaičius paslėptame sluoksnyje; $w_{i0} = b_i$, $W_i^T = (w_{i1}, w_{i2}, ..., w_{in})$ -$j$-tojo neurono svoriai; $f$- aktyvacijos funkcija.

Reikia pažymėti, kad aktyvacijos funkcija kiekvienam sluoksniui gali būti skirtinga. Bendras DSP išduodamas išėjimas $Y$ gali būti išreikštas tokiu būdu:

$$Y = f(V^T y + v_0)$$

čia $v_0 = b$, $V^T = (v_1, v_2, ..., v_k)$ -išėjimo neurono svoriai.

Daugiasluoksnių perceptronų skiriamieji paviršiai gali būti žymiai sudėtingesni nei vienasluoksnių, todėl jie ir yra populiaresni. Kiekvienas paslėpto sluoksnio neuronas apibrėžia hiperplokštumą, kuri klasifikuojamą erdvę padalina į dvi dalis. Paslėpto sluoksnio neurono hiperplokštumos lygtis:

$$\sum_{i=1}^{n} w_{ji} x_i + b_j = 0$$

kur *k* - neuronų skaičius paslėptame sluoksnyje. Išėjimo neuronas apjungia tas hiperplokštumas ir gaunasi žymiai sudėtingesnis skiriamasis paviršius.

### 1.2.1 Daugiasluoksnių perceptronų (DSP) apmokymas

Apmokant daugiasluoksnį perceptroną plačiai naudojama „backpropagation" metodika -tinklo apmokymas yra pradedamas nuo išėjimo sluoksnio, po to nuosekliai mokomi paslėpti sluoksniai. Perceptrono apmokymui yra būtina apmokymo duomenų - tinklo įėjimo vektorių *x* ir pageidaujamų rezultatų *t*, aibė :

$$\{x_1, t_1\}, \{x_2, t_2\}, ..., \{x_N, t_N\}.$$

Perceptrono optimalūs svoriai $w = (w_1, w_2, ..., w_L)$ yra parenkami taip, kad minimizuotų pasirinktą nuostolių funkciją - dažniausiai yra naudojama kvadratinė nuostolių funkcija (MSE):

$$E(w) = \frac{1}{N} \sum_{i=1}^{N} e_i^2 = \frac{1}{N} \sum_{i=1}^{N} (t_i - Y(x_i, w))^2$$

čia $Y$ tinklo generuojamas išėjimas, *N*- apmokymo duomenų aibės tūris.

Naudojant "backpropagation" metodiką aktyvacijos funkcija turi būti diferencijuojama. Daugiasluoksniuose tinkluose paslėptuose sluoksniuose dažniausiai naudojamos logistinės sigmoidinės funkcijos, kurios įėjimų reikšmių begalinę amplitudę suspaudžia į baigtinę išėjimų amplitudę. Sigmoidinės funkcijos nuolydis turi būti arti nulio, kai įėjimo reikšmė didelė. Logistinė funkcija yra gera tuo, jog turi gana paprastą išvestinės išraišką:
$f'(s) = af(s)(1 - f(s))$ ir yra diferencijuojama visoje abscisių ašyje.

Apmokant perceptronus populiariausi ir dažniausiai naudojami yra adaptaciniai iteraciniai nusileidimo metodai. Nusileidimo metodai yra pagrįsti informacija apie minimizuojamos MSE pirmąsias ir antrąsias dalines išvestines. Tokia informacija yra naudinga krypties link lokalaus minimumo parinkimui - funkcijos gradientas nukreiptas funkcijos greičiausio augimo kryptimi, todėl dažnai minimizuojant funkciją yra pasirenkama priešinga antigradiento kryptis.

Apmokymas pradedamas nuo pradinių svorių $w_0$ inicializacijos ir pasinaudojant informacija apie nuostolių funkcijos pirmąsias ir antrąsias dalines išvestines generuojama tolimesnė svorių seka.

Pačiu paprasčiausiu backpropagation algoritmo atveju DSP pradiniai inicializuoti svoriai $w_0$ keičiami tokiu iteraciniu būdu:

$$w_{k+1} = w_k + a_k d_k,$$

čia $w_k$- svoriai k-tosios iteracijos metu, $d_k = -g_k$ žingsnio kryptis, lygi antigradientui, $a_k$- žingsnio dydis, kuris paprasčiausiu atveju yra pastovus.

Gradientinio nusileidimo atveju nusileidimo žingsnis yra pastovus ir dažnai gana sunku jį parinkti. Kuo didesnė $a_k$ reikšmė, tuo greičiau DPS mokosi, tačiau jei apmokymo žingsnis yra per didelis, mokymas tampa nestabilus. Per mažas žingnis - algoritmas konverguoja lėtai - reikia daugiau iteracijų. Viena iš paprasčiausių žingsnio reguliavimo metodologijų, pagreitinančių algoritmo veikimą, yra tokia: jei minimizuojamos nuostolių funkcijos reikšmė iteracijų metu pastoviai mažėja, žingsnio dydį didinti, jei MSE ima didėti - žingsnį sumažinti.

Nors minimizuojama funkcija antigradiento kryptimi mažėja greičiausiai, tačiau žingsnis ta kryptimi nebūtinai garantuoja didžiausią konvergavimo greitį. Vienas iš būdų patobulinti gradientinį apmokymo algoritmą yra skirtas tam, kad apmokymas neįstrigtų nežymiuose lokaliuose MSE minimumuose:

$$d_k = -g_k + \beta_{k-1} d_{k-1}:$$

Pasirenkant žingsnio kryptį inercinio dėmens $\beta_{k-1} d_{k-1}$ pagalba algoritmas leidžia iššokti iš lokalių MSE minimumų. Toks algoritmas pasižymi didesniu konvergavimo greičiu, nei paprastas gradientinis, tačiau kaip ir paprasto gradientinio apmokymo atveju, jis yra gana lėtas NN apmokymo metodas.

Naudojantis aukščiau aprašytų algoritmų metodika yra sukurtas žymiai efektyvesnis DSP apmokymo metodas - jungtinių krypčių metodas. Naudojant jungtinių krypčių metodus, pirmas žingsnis yra daromas antigradiento kryptimi $d_0 = -g_0$, ta kryptimi atliekama tiesinė optimalaus žingsnio $a_0$ dydžio paieška. Sekančios iteracijos metu pasirenkama kryptis yra jungtinė pradinei krypčiai:

$$d_k = -g_k + \beta_k d_{k-1}, \quad \text{k>0}$$

Daugumos jungtinių krypčių algoritmų metu žingsnis $a$ kiekvienos iteracijos metu yra perskaičiuojamas atliekant tiesinę optimalaus žingsnio paiešką:

$$a_k = \arg\min_{a>0} E(w_k + ad_k)$$

Naudojant Niutono metodus, optimalaus žingsnio dydis $a_k$ parenkamas pasinaudojant MSE antrosiomis išvestinėmis:

$$w_{k+1} = w_k - H_k^{-1} g_k \ ,$$

čia $H_k$ - MSE antrųjų dalinių išvestinių svorių $w_k$ atžvilgiu matrica (hesianas).

Niutono metodas lyginant su standartiniais gradientiniais apmokymo metodais, pasižymi žymiai didesniu konvergavimo greičiu, tačiau tik tokiu atveju, kai pradiniai svoriai $w_0$ yra sąlyginai arti optimalių, kitu atveju gana dažnai būna sunku paskaičiuoti hesiano atvirkštinę matricą, todėl plačiau yra naudojami kvaziniutono metodai. Naudojant kvaziniutono metodus, antrosios dalinės išvestinės nėra skaičiuojamos, hesianą aproksimuojant pirmosiomis dalinėmis MSE išvestinėmis. Minimizuojant kvadratinę nuostolių funkciją hesianas gali būti aproksimuojamas tokiu būdu:

$$H_k = J_k^T J_k$$

Tokiu atveju gradientas gali būti apskaičiuojamas taip:

$$g_k = J_k^T e_k$$

čia $J$- jakobianas, sudarytas iš tinklo klaidų $e_i$, $i$ =1,2,…,$N$ pirmųjų dalinių išvestinių svorių $w_k$ atžvilgiu.

Naudojantis standartine "backpropagation" technika jakobianas yra žymiai paprasčiau suskaičiuojamas nei hesianas. Naudojant Levenberg-Marąuardt kvaziniutono metodą, iteracinė apmokymo formulė atrodo taip:

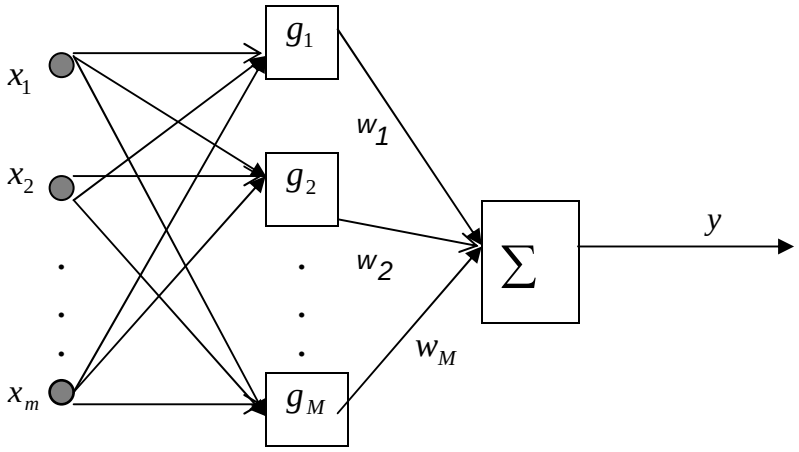
$$w_{k+1} = w_k - [J_k^T J_k + \mu_k I]^{-1} J_k^T e_k$$

Jei konstanta $\mu_k$ lygi nuliui, tai šis algoritmas tampa Niutono metodu su hesiano aproksimacija, didėjant $\mu_k$ reikšmei, algoritmas tampa gradientiniu su mažu žingsniu. Niutono metodas veikia gerai, kai svoriai yra arti optimalių, todėl po kiekvienos sėkmingos iteracijos $\mu_k$ yra mažinamas, priešingu atveju padidinamas.

## 2. Boosting search metodas

Duomenys:

Tarkime, kad turime poras $(\mathsf{x}_1, y_1),(\mathsf{x}_2, y_2),...,(\mathsf{x}_N, y_N)$, kur $\mathsf{x}_i = (x_{i1,} x_{i2},..., x_{im})$, $y_i$ - skaliariniai dydžiai.




Modelis : $y(x) = \sum_{i=1}^{M} w_i g_i(\vec{x}_i)$,

kur $w_i$-modelio svoriai; $g_i(x) = g\ (\vec{x} / \theta_i)$- duotos funkcijos, priklausančios nuo nežinomų $\theta_i$.

Pažymėkime $\begin{cases} y_i^{(0)} = y_i \\ y_i^{(k)} = y_i^{(k-1)} - w_k g_k(\mathsf{x}_i) \end{cases}$, $1 \le i \le N$

ir $MSE_k = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k)})^2$ - kvadratinių paklaidų vidurkis, kai y aproksimuojame pirmomis k bazinėmis funkcijomis.

Išveskime išreikštinę $MSE_k$ pavidalą.

Jei k=1, tai $MSE_1 = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(1)})^2 = \frac{1}{n}\sum_{i=1}^{N}(y_i^{(0)} - w_1 g_1(\mathsf{x}_i))^2$

Jei k=2, tai

$$MSE_2 = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(2)})^2 = \frac{1}{n}\sum_{i=1}^{N}(y_i^{(1)} - w_2 g_2(\mathsf{x}_i))^2 = \frac{1}{n}\sum_{i=1}^{N}(y_i - w_1 g_1(\mathsf{x}_i) - w_2 g_2(\mathsf{x}_i))^2 =$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{2} w_j g_j(\mathsf{x}_i))^2$$

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

Bendru atveju

$$MSE_k = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k)})^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)} - w_k g_k(\mathsf{x}_i))^2 = \frac{1}{N}\sum_{I=1}^{N}(y_i^{(k-2)} - w_{k-1} g_{k-1}(\mathsf{x}_i) -$$

$$- w_k g_k(\mathsf{x}_i))^2 = \frac{1}{N}\sum_{I=1}^{N}(y_i^{(k-3)} - w_{k-2} g_{k-2}(\mathsf{x}_i) - w_{k-1} g_{k-1}(\mathsf{x}_i) - w_k g_k(\mathsf{x}_i))^2 = \ldots =$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{k} w_j g_j(\mathsf{x}_i))^2 .$$

Tam, kad galėtume taikyti modelį , reikia rasti $w_k$ ir $\theta_k$ įverčius.
Tuo tikslu įsivedama empirinė koreliacijos funkcija

$$C_k(\theta_k) = \frac{\sum_{i=1}^{N} g_k(\mathsf{x}_i) y_i^{(k-1)}}{\sqrt{\sum_{i=1}^{N} g_k^2(\mathsf{x}_i)}\sqrt{\sum_{i=1}^{N}(y_i^{(k-1)})^2}},$$

kuri nusako ryšį tarp funkcijos $g_k(\mathsf{x})$ ir y. Kitaip sakant laikome, kuo didesnis $|C_k(\theta_k)|$ modulis, tuo geriau $g_k$ prognozuoja y. Todėl $\theta_k$ įverčius randame maksimizuodami $|C_k(\theta_k)|$.
Kai turime $\theta_k$ įverčius, tai $w_k$ įvertis randamas mažiausių kvadratų metodu:

$$MSE_k = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k)})^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)} - w_k g_k(\mathsf{x}_i))^2 \qquad (2.1)$$

$$\frac{\partial MSE_k}{\partial w_k} = -2\frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)} - w_k g_k(\mathsf{x}_i)) g_k(\mathsf{x}_i)$$

$$-2\frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)} - w_k g_k(\mathsf{x}_i)) g_k(\mathsf{x}_i) = 0 \quad \Rightarrow \quad \widehat{w}_k = \frac{\sum_{i=1}^{N} y_i^{(k-1)} g_k(\mathsf{x}_i)}{\sum_{i=1}^{N} g_k^2(\mathsf{x}_i)}$$

Įstatę $\widehat{w}_k$ į formulę (2.1) gaunam, kad $MSE_K =$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)} - w_k g_k(\mathsf{x}_i))^2 = \frac{1}{N}\sum_{i=1}^{N}\left( y_i^{(k-1)} - g_k(\mathsf{x}_i)\frac{\sum_{j=1}^{N} y_j^{(k-1)} g_k(\mathsf{x}_j)}{\sum_{j=1}^{N} g_k^2(\mathsf{x}_j)} \right)^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)})^2 -$$

$$- \frac{1}{N}\sum_{i=1}^{N}\left( \frac{2y_i^{(k-1)} g_k(\mathsf{x}_i)\sum_{j=1}^{N} y_j^{(k-1)} g_k(\mathsf{x}_j)}{\sum_{j=1}^{N} g_k^2(\mathsf{x}_j)} \right) + \frac{1}{N}\sum_{i=1}^{N}\left( \frac{g_k(\mathsf{x}_i)^2\left(\sum_{j=1}^{N} y_j^{(k-1)} g_k(\mathsf{x}_j)\right)^2}{\sum_{j=1}^{N} g_k^2(x_j)^2} \right) =$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)})^2\left( 1 - \frac{2\sum_{i=1}^{N} y_i^{(k-1)} g_k(\mathsf{x}_i)\sum_{j=1}^{N} y_j^{(k-1)} g_k(\mathsf{x}_j)}{\sum_{j=1}^{N} g_k^2(\mathsf{x}_j)\sum_{i=1}^{N}(y_i^{(k-1)})^2} + \frac{\sum_{i=1}^{N} g_k(\mathsf{x}_i)^2\left(\sum_{j=1}^{N} y_j^{(k-1)} g_k(\mathsf{x}_j)\right)^2}{\sum_{i=1}^{N}(y_i^{(k-1)})^2\left(\sum_{j=1}^{N} g_k^2(\mathsf{x}_j)^2\right)} \right) =$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)})^2\left( 1 - \frac{\sum_{i=1}^{N} y_i^{(k-1)} g_k(\mathsf{x}_i)^2}{\sum_{i=1}^{N} g_k^2(\mathsf{x}_j)\sum_{i=1}^{N}(y_i^{(k-1)})^2} \right) \Rightarrow$$

$$MSE_k = \frac{1}{N}\sum_{i=1}^{N}(y_i^{(k-1)})^2(1 - C_k^2(\theta_k))$$

Taigi matome, kad $MSE_K$ minimizavimas pagal $\theta_k$ parametrus ekvivalentus $|C_k(\theta_k)|$ maksimizavimui pagal $\theta_k$.

## 2.1 Skaičiavimo algoritmas

Inicializacija

Pažymėkime $u_k = \theta_k$. Tada algoritmo schema yra tokia.

Priskiriame t=0 ir pasirenkame fiksuotą skaičių s. Atsitiktiniai sugeneruojame s parametro realizacijų $u_k^{(1)}(t), u_k^{(2)}(t), ..., u_k^{(s)}(t)$ ir kiekvienai sugeniruotai reikšmei priskiriame svorį $\delta_i(t) = \frac{1}{s}$, i =1,2,..,s. Pasirenkame maksimalų iteracijų skaičių $M_I$ ir $\xi_b > 0$, skirtus nuspręsti, kada sustoti.

Žingsnis 1 (boosting)

1.1 Paskaičiuojame $\mathrm{cost}_j$ =1-$\left|C_k(u_k^{(j)}(t)\right|$, j=1,2,…,s

1.2 Randome tokį $u_k^{(j)}(t)$, kuriam 1-$\left|C_k(u_k^{(j)}(t)\right|$ mažiausias, ir tokį kuriam 1-$\left|C_k(u_k^{(j)}(t)\right|$, didziausias, t.y.

$$u_k^{(best)}(t) = \arg\min\{\mathrm{cost}_j, 1 \le j \le s\}$$

$$u_k^{(worst)}(t) = \arg\max\{\mathrm{cost}_j, 1 \le j \le s\}.$$

1.3 Normalizuojame nuostolius $\mathrm{loss}_j = \frac{\mathrm{cost}_j}{\sum_{l=1}^{s} \mathrm{cost}_l}$, kur j=1,2,...,s

1.4 Paskaičiuojame svorių faktorių $\beta_t$, naudodami formules

$$\xi_t = \sum_{j=1}^{s} \delta_j(t) loss_j; \quad \beta_t = \frac{\xi_t}{(1-\xi_t)}$$

1.5 Paskaičiuojame svorius $\delta_j(t+1) = \begin{cases} \delta_j(t)\beta_t^{loss_j}, \beta_t \le 1 \\ \delta_j(t)\beta_t^{1-loss_j}, \beta_t > 1 \end{cases}$, kur j=1,2,…,s

1.6 Normalizuojame svorius $\delta_j(t+1) = \frac{\delta_j(t+1)}{\sum_{l=1}^{s} \delta_l(t+1)}$, kur j=1,2,…,s.

Žingsnis 2

2.1 Sukonstruojame (s+1) tašką $u_k^{(s+1)}(t) = \sum_{i=1}^{s} \delta_i(t+1) u_k^{(i)}(t)$

2.2 Sukonstruojame (s+2) tašką $u_k^{(s+2)} = u_k^{(best)}(t) + (u_k^{(best)}(t) - u_k^{(s+1)}(t))$

2.3 $u_k^{worst}(t)$ pakeičiame geresniu iš $u_k^{(s+1)}(t)$ ir $u_k^{(s+2)}(t)$ (geresnis tas, kuriam cost mažesnis)

t=t+1. Jei t didesnis negu maksimalus iteracijų skaičius $M_I$ ( t> $M_I$ ) arba

$\left\| u_k^{(s+1)}(t) - u_k^{(s+1)}(t-1) \right\| < \xi_b$ baigiam darbą, jei ne, einam į žingsnis 1.

$u_k = u_k^{best}(t)$

Tam, kad gautume geresnį $\theta_k$ įvertį, algoritmas prasukamas keletą kartų pagal tokią schemą:

Inicializacija:

Parenkame iteracijų skaičių $M_R$ ir $\xi_r$ >0, skirtus sustojimui reguliuoti.

Pirmas žingsnis:

Atsitiktinai sugeneruojame s realizacijų $u_k^{(1)}, u_k^{(2)}, ..., u_k^{(s)}$ ir iškvietę boosting search gauname $u_k^{best}(0)$.

Ciklas:

for l=1 to $M_R$

$u_k^{(1)} = u_k^{best}(l-1)$

atsitiktinai sugeneruoti dar (s-1) taške $u_k^{(i)}$, i=1,2,…,s

iškviesti boosting seach ir gauti $u_k^{best}(l)$

if $\left\| u_k^{best}(l-1) - u_k^{best}(l) \right\| < \xi_r$ then

išeiti iš ciklo

end

end

$u_k = u_k^{best}(l)$

# 3. Entropija

Skaitinis matas, parodantis atsitiktinio dydžio atsitiktinumo lygį, vadinamas entropija.

Jei X absoliučiai tolydus a.d., tai jo entropija apibrėžiama taip : $H(p) = -\int p(x)\log p(x)dx$ .

Ji vadinama Klasikinė Shannon entropija. Laikoma $\log p(x)$=0, kai $p(x)$=**0.**

Alternatyvus entropijos variantas- q-Renyi entropija. Kai $q \neq 1$ ji apibrėžiama taip:

$H_q(p) = \frac{1}{1-q}\log\left(\int p(x)^q dx\right)$, q>0

Kai $q \to 1$, $H_q(p)$ konverguoja į klasikinę Shannon entropiją.

Ištikrųjų,

pažymėkime $f(q) = \log\left(\int p(x)^q dx\right)$, o $g(q) = 1-q$ ir pritaikykime Liopitalio taisyklę funkcijos $H_q(p) = \frac{f(q)}{g(q)}$ ribai paskaičiuoti.

Turime, kad

$$\left[\log\int p(x)^q dx\right]_q' = \frac{1}{\int p(x)^q dx}\left[\int p(x)^q dx\right]_q'$$

$$\left[\int p(x)^q dx\right]_q' = \int\left[p(x)^q\right]_q' dx = \int p(x)^q \log p(x)dx$$

Iš kur gauname,

$$\lim_{q\to 1} H_q(p) = \lim_{q\to 1}\frac{f(q)}{g(q)} = \lim_{q\to 1}\frac{f'(q)}{g'(q)} = \lim_{q\to 1}\frac{-\int p(x)^q \log p(x)dx}{\int p(x)^q dx} =$$

$$= -\int p(x)\log p(x)dx = H(p)$$

## 3.1 Tankis maksimizuojantis entropiją

Tarkime, kad a.v. $X = (X_1, X_2, ..., X_n)$ tankis yra Gauso su $\mu$ ir $\sum$ parametrais, t.y.

$$G(\mathsf{x};\mu;\sum)=\frac{1}{((2\pi)^{\frac{n}{2}}\left|\sum\right|^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}(\mathsf{x}-\mu)^{\mathrm{T}}\sum{}^{-1}(\mathsf{x}-\mu)\right\}$$

Straipsnyje [3] parodyta, kad tankis $G(\mathsf{x};\mu;\sum)$ maksimizuoja Shannon entropiją. Toliau nagrinėsime Renyi entropiją.

Su $q>\frac{n}{n+2}$ ir $q\neq 1$ apibrėžkime n-matį tankį $f_{q,C}$ taip:

$$f_{q,C}(\mathsf{x})=A_q(1-(q-1)\beta\mathsf{x}^T C^{-1}\mathsf{x})_+^{\frac{1}{q-1}},$$

kur $\beta=\beta_q=\frac{1}{2q-n(1-q)}$ ir normalizuojanti konstanta

$$A_q=\begin{cases}\left(\Gamma\left(\frac{1}{q-1}\right)(\beta(1-q))^{\frac{n}{2}}\right)\Big/\left(\Gamma\left(\frac{1}{q-1}-\frac{n}{2}\right)\pi^{\frac{n}{2}}|C|^{\frac{1}{2}}\right), & \frac{n}{n+2}.<q<1\\ \left(\Gamma\left(\frac{q}{q-1}+\frac{n}{2}\right)(\beta(q-1))^{\frac{n}{2}}\right)\Big/\left(\Gamma\left(\frac{q}{q-1}\right)\pi^{\frac{n}{2}}|C|^{\frac{1}{2}}\right), & q>1\end{cases}$$

kur $x_+=\max(x;0)$.

Šio tankio apibrėžimo sritį žymėsime $\Omega_{q,\mathsf{C}}$ duotam $f_{q,C}(\mathsf{x})$, kai $q>1$ lygus

$\Omega_{q,\mathsf{C}}=\left\{\mathsf{x}:\mathsf{x}^T\mathsf{C}^{-1}\mathsf{x}\le\frac{2q}{q-1}+n\right\}$, o jei $q<1$, tai $\Omega_{q,\mathsf{C}}=\mathbf{R}^n$.

Kadangi $\lim_{q\to 1}\frac{\Gamma\left(\frac{1}{1-q}\right)(1-q)^{\frac{n}{2}}}{\Gamma\left(\frac{1}{1-q}-\frac{n}{2}\right)}=1$, tai

$$\lim_{q\to 1}(1-(q-1)\beta\mathsf{x}^T\mathsf{C}^{-1}\mathsf{x})_+^{\frac{1}{q-1}}=\exp\left(-\mathsf{x}^T\mathsf{C}^{-1}\frac{\mathsf{x}}{2}\right).$$

Įš tikrųjų.

Pažymėkime $\beta\mathsf{x}^T\mathsf{C}^{-1}\mathsf{x}=$a,

tada $\lim_{q\to 1}(1-(q-1)\beta \mathsf{x}^T \mathsf{C}^{-1}\mathsf{x})_+^{\frac{1}{q-1}} = \lim_{q\to 1}(1-(q-1)a)_+^{\frac{1}{1-q}} = [q-1=b] =$

$$= \lim_{b\to 0}(1-ba)_+^{\frac{1}{b}} = \lim_{b\to 0}\exp\ln(1-ab)_+^{\frac{1}{b}} = \lim_{b\to 0}\exp\frac{1}{b}\big(-ab+O(b)\big) = \exp a =$$

$$= \exp\left(-\mathsf{x}^T\mathsf{C}^{-1}\frac{\mathsf{x}}{2}\right)$$

Todėl riba $\lim_{q\to 1} f_{q,\mathsf{C}}(\mathsf{x}) = f_{1,\mathsf{C}}(\mathsf{x}) = ((2\pi)^n|\mathsf{C}|)^{-\frac{1}{2}}\exp\left(-\mathsf{x}^T\mathsf{C}^{-1}\frac{\mathsf{x}}{2}\right)$ yra Gauso.

Reiškia $f_{q,C}$ maksimizuoja ir $H(p)$ entropiją.

# 4. Klasifikavimas

Objektų klasifikavimo reikia daugelyje mokslinės ir praktinės veiklos sričių. Klasifikavimas pastoviai naudojamas gamyboje. Priimamosios, tarpiniuose, bei išleidžiamosios kontrolės punktuose gaminiai yra skirstomi į gerus ar blogus, remontuotinus ir pan. Taip pat kontroliuojami technologiniai procesai ir jų režimai ir priimami sprendimai apie jų normalų funkcionavimą arba išsiderimą.

Pagal nusistovėjusią praktiką sprendimai apie objektų priskyrimą vienai ar kitai kategorijai įvairiuose kontrolės punktuose paprastai atliekami lyginant gautuosius parametrų matavimus su leistinomis ribomis, nurodytomis techninėje dokumentacijoje.

Sprendimai būtų tikslesni, jeigu jie būtų grindžiami visais parametrų matavimais, gautais pastarajame ir ankstesniuose kontrolės punktuose. Ypač tuo atveju, kai gaminiai yra sudėtingi, o parametrai priklausomi. Šiuolaikiniai kompiuteriniai tinklai ir automatizuotos duomenų įrašymo ir perdavimo sistemos sudaro galimybę realiame laiko mastelyje priimant sprendimą bet kuriame kontrolės punkte, panaudoti informaciją iš ankstesnių kontrolės punktų.

Kineskopų gamykloje kiekvienam kineskopui priskiriamas identifikacinis numeris,atliekant technologines ir kontroles operacijas duomenų bazėse yra fiksuojama kiekvieno kineskopo istorija,.t.y. technologinių režimų, prie kurių jis buvo pagamintas, charakteristikos, kineskopo parametrų kontrolinėse operacijose reikšmės, bei rodikliai.

Remdamasi sukaupta informacija galima atskirti kineskopus, kurie potencialiai turi daugiau šansų patekti į brokuotų kineskopų kategoriją.

## 4.1 Klasifikavimo tikslumas

Tarkime, kad norime prognozuoti kintamąjį Y, įgyjanti dvi reikšmes 0 ir 1, remiantis atsitiktinio vektoriaus $\mathbf{X} = (X_1, X_2, ..., X_N)$ matavimais.

Tegu $Y = \begin{cases} 1, \text{ jeigu gaminys geras,} \\ 0, \text{ jeigu gaminys brokuotas.} \end{cases}$

Tarkime klasifikavimo slenksnis yra z, tada klasifikavimo taisyklę galima suformuluoti taip:

-gaminys yra geras, jei output≥z;

-gaminys yra brokuotas, jei output<z.

Modelio tinkamumą objektų klasifikavimui galima apibūdinti klasifikavimo lentele, kuri parodo, kaip tiksliai sukonstruotas klasifikatorius atskiria gerus ir blogus gaminius.

Įveskime atsitiktinį dydį $\eta$, kuris įgyja reikšmę 1, jeigu gaminys priskirtas gerų kategorijai, ir įgyja reikšmę 0, jeigu gaminys priskirtas brokuotų kategorijai.

Apibrėžkime sąlygines tikimybes

$$\alpha_{ij} = \mathbf{P}\{\eta = i \mid Y = j\}, \quad \text{i, j=0,1.}$$

$\alpha_{00}$ ir $\alpha_{11}$ yra teisingi sprendimai, $\alpha_{10}$ ir $\alpha_{01}$ - klaidingi. Klasifikavimo taisyklė tuo geresnė, kuo didesnės tikimybės $\alpha_{00}$ ir $\alpha_{11}$, ir kuo mažesnės tikimybės $\alpha_{10}$ ir $\alpha_{01}$.

Praktikoje svarbesnės yra atvirkštinės tikimybės

$$\beta_{ji} = \mathbf{P}\{Y = j \mid \eta = i\} = \frac{\alpha_{ij} w_j}{\alpha_{i1} w_1 + \alpha_{i0} w_0}, \text{ kur } i,j = 0,1. \qquad (4.1.1)$$

kurios parodo gaminių srautų, gautų atlikus klasifikavimą, užterštumą kitos grupės objektais. $w_0$ ir $w_1$ yra apriorinės tikimybės :

$$w_0 = \mathbf{P}\{Y=0\}, \qquad w_1 = \mathbf{P}\{Y=1\}, \qquad w_0 + w_1 = 1$$

Defektiniais pripažintų gaminio srauto dydis

$$Q = \mathbf{P}\{\eta = 0\} = \alpha_{01} w_1 + \alpha_{00} w_0 \qquad (4.1.2)$$

Jeigu defektiniais pripažintų gaminių dalis $w_1$ yra maža, tai įrašę formulėje (4.1.1) ir (4.1.2) vardikliuose $w_0 = 0$ ir $w_1 = 1$, gausime apytiksles formulas

$$Q \approx \alpha_{01}, \qquad \beta_{01} \approx w_0 \frac{\alpha_{10}}{\alpha_{11}} = w_0 \gamma, \qquad \beta_{00} \approx w_0 \frac{\alpha_{00}}{\alpha_{01}} = w_0 \delta \qquad (4.1.3)$$

kurių pagalba galime apibūdinti klasifikavimo tikslumą, net ir nežinodami tikslių tikimybių $w_0$ ir $w_1$ reikšmių. . Būtent, gerais pripažintų gaminių sraute defektingumo lygis apytiksliai sumažėja nuo $w_0$ iki $w_0 \gamma$, o blogais pripažintų gaminių sraute- padidėja nuo $w_0$ iki $w_0 \delta$.

Praktikoje tikimybės $\alpha_{ij}$, $i,j=0,1$, yra nežinomos ir jas reikia vertinti iš statistinių duomenų.

Pažymėkime $V_{11}$ ir $V_{01}$ gerais ir defektiniais pripažintų gaminių skaičių gerų gaminių aibėje ($V_{11} + V_{01} = N_1$); analogiškai $V_{10}$ ir $V_{00}$ - brokuotų gaminių aibėje ($V_{10} + V_{00} = N_0$)

| $Y/\eta$ | 1 | 0 | $\sum$ |
|---|---|---|---|
| 1 | $V_{11}$ | $V_{01}$ | $N_1$ |
| 0 | $V_{10}$ | $V_{00}$ | $N_0$ |

lentelė 1. Klasifikavimo rezultatai

Naudodami šiuos duomenis gauname tikimybių $\alpha_{ij}$ įverčius

$$\hat{\alpha}_{ij} = \frac{V_{ij}}{N_j}, \quad i,j=0,1.$$

Klasifikavimo rezultatus charakterizuojantys skaičiai $V_{ij}$ pateikti 1.lentelėje, o tikimybių $\alpha_{ij}$ įverčiai $\hat{\alpha}_{ij}$ - lentelėje 2.

| $Y/\eta$ | 1 | 0 | $\sum$ |
|---|---|---|---|
| 1 | $\hat{\alpha}_{11}$ | $\hat{\alpha}_{01}$ | 1 |
| 0 | $\hat{\alpha}_{10}$ | $\hat{\alpha}_{00}$ | 1 |

Lentelė 2. Tikimybių $a_{ij}$ įverčiai

Turėdami įverčius $\hat{\alpha}_{ij}$ gauname charakteristikų ( 4.1.3) įverčius

$$\hat{Q} = \hat{\alpha}_{01}, \qquad \hat{\gamma} = \frac{\hat{\alpha}_{10}}{\hat{\alpha}_{11}}, \qquad \hat{\delta} = \frac{\hat{\alpha}_{00}}{\hat{\alpha}_{01}}$$

# 5. Eksperimentų aprašymas

Buvo nagrinėjimas kineskopų klasifikavimo uždavinys, turint kineskopo parametrų reikšmes (st_i_vakum, st_vid_i_kat, st_u_koef; st_i_koef, st_i_vakuumo1, st_vid_i, st_i_koef_2, st_u_koef_2 ).

Tikslas – žinant parametrų reikšmes nuspręsti ar kineskopas brokuotas, ar geras.

Pažymėkime

$$Y = \begin{cases} 0, \text{ jeigu kineskopas geras,} \\ 1, \text{ jeigu kineskopas brokuotas.} \end{cases}$$

Šiam uždaviniui spręsti pasirinkti neoroniniais tinklai, o tiksliau boosting search metodas. Vienu atveju buvo naudojama Gauso aktyvacijos funkcija, kitu- Renyi entropiją maksimizuojatis tankis.

Turimi stebėjimai (6695 gerų kineskopų) ir (303 brokuotų kineskopų) buvo suskirstyti į dvi dalis:
viena dalis naudojama parametrų įverčiams rasti, o kita dalis tarnauja kaip testinė aibė klasifikavimo tikslumui įvertinti sudarant klasifikavimo lentelę.

Eksperimento metų buvo bandoma parinkti optimalią tinklo architektūrą. Buvo keičiamos aktyvacijos funkcijų skaičius, tačiau didinant funkcijų skaičių prognuozės tikslumas ženkliai negėrėja, todėl pateikiamos klasifikavimo lentelės buvo gautos naudojant 3 aktyvacijos funkcijas. Optimalaus tinklo paieška buvo pradėta nuo tinklo su atsitiktinai pasirinktomis nežinomų reikšmių vektoriais $(u_1, u_2, u_3)$, kur $u_i$ žymi i-os funkcijos nežinomų parametrų vektorių. Nežinomų parametrų įverčiai pateikiami priedas1. Eksperementai realizuoti su Matlab 7.0 (priedas 2)

Modelio tinkamumas objektų klasifikavimui apibūdintas klasifikavimo lentelių pagalba, iš kurių matoma kiek tiksliai modelis atskiria gerus ir brokuotus kineskopus.

## 5.1 Kineskopų klasifikavimo rezultatai

### 5.1.1 Įprastos klasifikavimo lentelės

Tam, kad pailiustruoti klasifikavimo taisyklės veikimą, parinkome keletą klasifikavimo slenkčių ir sudarėme jiems klasifikavimo lenteles, taip pat radome tokių tikimybių kaip $\alpha_{11}$ (

tikimybė, kad gaminys pripažintas geru, jei jis geras) ir $\alpha_{00}$ ( tikimybė, kad gaminys pripažintas brokuotu, jei jis brokuotas) įverčius.

Žemiau pateikiamos klasifikavimo lentelės imant skirtingus slenksčius. Kiekvienam slenksniui sudaromos po dvi lenteles ( pirma lentelė- klasifikavimo rezultatai, antra- tikimybių $a_{ij}$ įverčiai). Pirmas lentelių rinkinys gautas naudojant aktyvacijos funkcijas maksimizuojančias Renyi entropiją, antras- naudojant Gauso aktyvacijos funkcijas.

$c_1$=0.4

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3321 | 28 | 3349 |
| Brokuotas | 132 | 18 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.99 | 0.008 | 1 |
| Brokuotas | 0.88 | 0.12 | 1 |

$c_2$=0.5

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3307 | 40 | 3349 |
| Brokuotas | 122 | 28 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.988 | 0.012 | 1 |
| Brokuotas | 0.81 | 0.19 | 1 |

$c_3$=0.6

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3287 | 62 | 3349 |
| Brokuotas | 109 | 41 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.982 | 0.018 | 1 |
| Brokuotas | 0.726 | 0.273 | 1 |

$c_4$=0.7

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3240 | 109 | 3349 |
| Brokuotas | 91 | 59 | 150 |

| | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Geras | 0.967 | 0.03 | 1 |
| Brokuotas | 0.61 | 0.39 | 1 |

Matome, kad įvairiems klasifikavimo slenksčiams neblogi rezultatai atpažįstant gerus kineskopus. Tuo tarpu blogi kineskopai klasifikuojami ne taip tiksliai. Antra vertus knygoje [1], lentelėje 7.14 randame, kad sprendžiant panašaus tipo uždavinius rezultatai gaunami panašūs (palyginti su lentele, atitinkančia c=0,6 pirmu atveju, ir c=0,45 antru atveju).

Dabar pateiksime rezultatus antru atveju, analogiškai kaip ir pirmu atveju imsime skirtingus slenksčius:

$c_1$=0.4

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3305 | 44 | 3349 |
| Brokuotas | 113 | 37 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.986 | 0.013 | 1 |
| Brokuotas | 0.75 | 0.246 | 1 |

$c_2$=0.45

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3288 | 61 | 3349 |
| Brokuotas | 108 | 42 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.982 | 0.018 | 1 |
| Brokuotas | 0.72 | 0.28 | 1 |

$c_3$=0.5

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3274 | 75 | 3349 |
| Brokuotas | 103 | 47 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.978 | 0.022 | 1 |
| Brokuotas | 0.69 | 0.31 | 1 |

$c_4$=0.55

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Geras | 3256 | 93 | 3349 |
| Brokuotas | 96 | 54 | 150 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.972 | 0.027 | 1 |
| Brokuotas | 0.64 | 0.36 | 1 |

Matome, kad tiek naudojant pirmojo tipo aktyvacijos funkcijas, tiek naudojant Gauso aktyvacijos funkcijas klasifikavimo rezultatai panašūs. Toks rezultatas nėra netikėtas, kadangi skyrelyje 4., buvo parodyta, kad pirmojo tipo aktyvacijos funkcija yra Gauso, kai q=1.

### 5.1.2 Modifikuotos klasifikavimo lentelės

Kaip ir skyrelyje 5.1.1 naudojame abiejų tipų aktyvacijos funkcijas. Tačiau, šiuo atveju elgėmės taip. Pasirinkome skaičių a ir b : $0<a<b<1$. Jei tinklo išėjimas $y<a$, tai laikėme, kad gaminys pripažintas blogu, jei $y>b$ - geru. Jeigu $y \in (a, b)$, tai toks kineskopas laikomas įtartinu ir sudarant klasifikavimo lentelę jis neįtraukiamas.

Pirmu atveju gauname:

Jei (a,b)=(0.6;0.7)

Įtartinų kineskopų 65 (18 blogų ir 47 gerų)

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3240 | 62 | 3302 |
| Brokuotas | 91 | 41 | 132 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.981 | 0.018 | 1 |
| Brokuotas | 0.69 | 0.31 | 1 |

Jei (a,b)= (0.6; 0.8)

Įtartinų kineskopų 215 (36 blogų ir 179 gerų).

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3108 | 62 | 3170 |
| Brokuotas | 73 | 41 | 114 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.98 | 0.02 | 1 |

| | | | |
|---|---|---|---|
| Brokuotas | 0.64 | 0.36 | 1 |

Jei (a,b)=(0.6; 0,85),

Įtartinų kineskopų 393 (346 gerų ir 47 blogų kineskopų)

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 2941 | 62 | 3003 |
| Brokuotas | 64 | 41 | 103 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.98 | 0.02 | 1 |
| Brokuotas | 0.6 | 0.4 | 1 |

Antru atveju :

Jei (a,b)= (0,4;0,6)

Įtartinų kineskopų 108 (27 blogų ir 81 gerų)

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3224 | 44 | 3268 |
| Brokuotas | 86 | 37 | 123 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.987 | 0.013 | 1 |
| Brokuotas | 0.7 | 0.3 | 1 |

Jei (a,b)=(0,45;0,65)

Įtartinų kineskopų 126 (99 gerų, 29blogų)

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3189 | 61 | 3250 |
| Brokuotas | 79 | 42 | 121 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.981 | 0.019 | 1 |
| Brokuotas | 0.65 | 0.35 | 1 |

Jei (a,b)= (0,45;0,75)

Įtartinų kineskopų -273 (232gerų ir 41 blogų)

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 3056 | 61 | 3117 |

| | | | |
|---|---|---|---|
| Brokuotas | 69 | 42 | 109 |

| *Y/η* | Pripažintas geru | Pripažintas brokuotu | Viso |
|---|---|---|---|
| Geras | 0.98 | 0.019 | 1 |
| Brokuotas | 0.61 | 0.39 | 1 |

Kartojant bandymus, buvo pastebėta, kad pirmu atveju paėmus intervalą (0.6; 0.8), o antru (0.45;0.75) didelis kineskopų skaičius pripažįstamas įtartinais, tačiau labai pagerėja klasifikavimo taisyklė.

Matome, kad įvairiems klasifikavimo slenksčiams neblogi rezultatai atpažįstant gerus kineskopus. Bet brokuoti kineskopai klasifikuojami ne taip tiksliai. Įtartinus kineskopus ir pripažintus brokuotais reikia pertikrinti arba persiklasifikuoti kitais metodais.

## Išvados

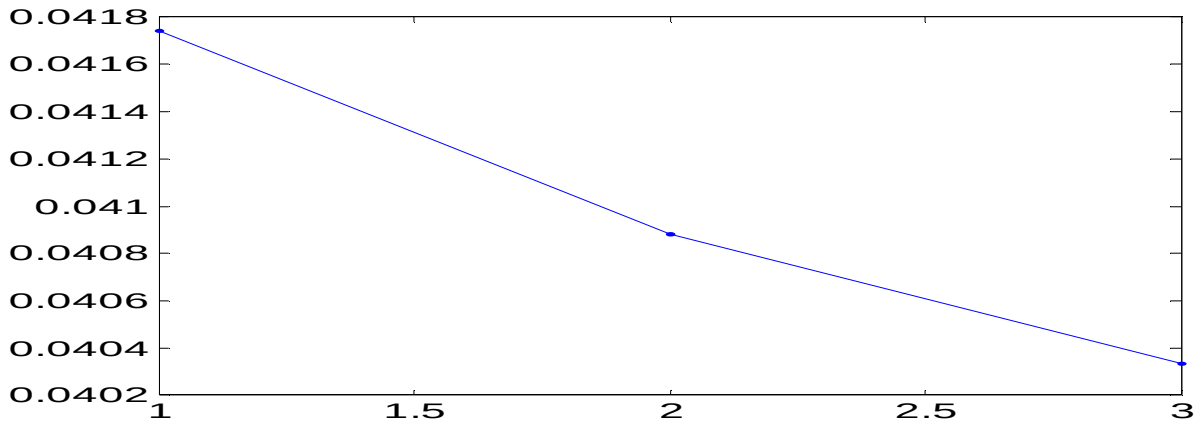
Darbe nagrinėtas atskiras neuroninio tinklo modelis, naudojantis Gauso aktyvacijos funkcijas ir funkcijas, maksimizuojančias Renyi entropiją. Modelis naudotas spręsti klasifikavimo uždavinį. Gauti rezultatai panašūs į egzistuojančių modelių rezultatus. Galima teigti, kad modelis tinkamas spręsti praktinius uždavinius.

## Priedas 1

Įvertintų parametrų rezultatai

Pirmu atveju

MSE reikšmės : 0.0417 0.0409 0.0403




MSE grafikas

Svoriai 1,1075 -20,992 1,6935

Pirmas regresorius

Kovariacijos matrica

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17.5716 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 25.7139 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 30.3895 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 11.9254 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 16.9579 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 19.6360 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 5.5006 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11.7564 |

Vidurkių vektorius

-0.2520 0.3800 -0.5321 0.4838 0.1442 -0.5473 0.2677 0.0338

Antras regresorius

Kovariacijos matrica

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11.6501 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 9.3799 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 12.4900 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 10.3985 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 10.3674 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 11.3099 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 11.8539 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10.1348 |

Vidurkių vektorius

-3.9727 11.0303 3.7992 0.7765 -3.0981 0.3699 -0.3532 3.6647

Trečias regresorius

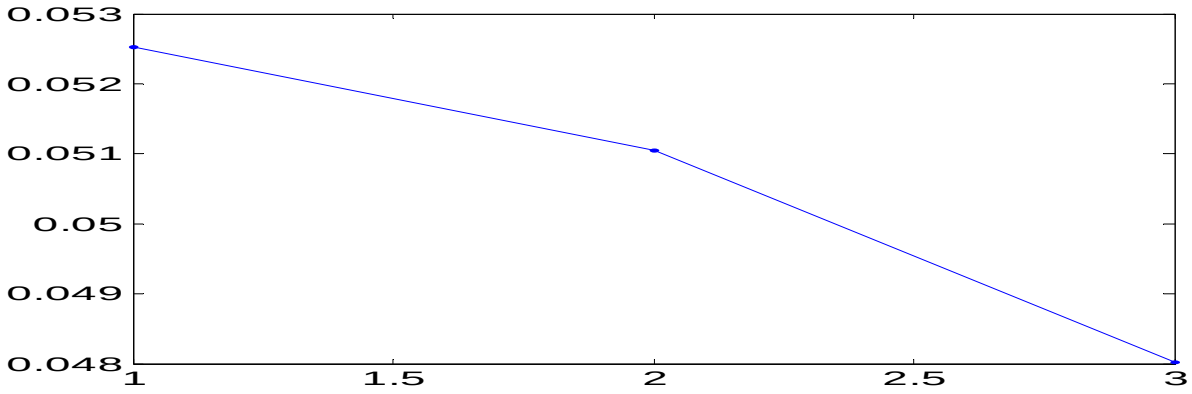
Kovariacijos matrica

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 9.4035 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 12.1551 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 9.9316 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 11.4465 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 10.4140 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 9.0484 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 12.5192 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6.6908 |

Vidurkių vektorius

-5.9300 -1.0218 -1.7977 5.8882 -5.4335 -2.8082 5.2073 0.9182

Antru atveju



Mse grafikas

| | | | |
|---|---|---|---|
| Mse | 0.0525 | 0.0511 | 0.0480 |
| Svoriai | 1,224 | 0,34148 | -0,13001 |

Pirmas regresorius

Kovariacijos matrica

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16.8513 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 15.2933 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 19.1513 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 23.0502 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 7.4458 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 18.3681 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 19.2929 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7.6600 |

Vidurkių vektorius

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Antras regresorius

Kovariacijos matrica

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 12.0580 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 7.5587 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 8.4937 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 9.4117 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 9.6073 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 10.1466 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 12.4762 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8.4264 |

Vidurkių vektorius

| | | | | | | |
|---|---|---|---|---|---|---|
| -1.5896 | -3.3475 | 3.5824 | 2.5976 | -1.8465 | -1.3581 | -0.4437 |

Trečias regresorius

Kovariacijos matrica

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 20.4495 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 10.6965 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0.1933 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 6.6342 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 6.3910 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 14.1515 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 11.9103 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12.9956 |

Vidurkių vektorius 0 0 0 0 0 0 0 0

## Priedas 2

Programos kodas

boost_kernel_output

```
function out = boost_kernel_output(kernel,input)
    q = 1.2;
    n = length(input);
    beta = 1/(2*q - n*(1-q));
    out = (1 - (q - 1)*beta*((input - kernel.mean)*inv(kernel.covariance)*(input - kernel.mean)'));
    out = max(out,0);
    out = out^(1/(q-1));


function out = boost_kernel_output(kernel,input)
     out = exp ( - (1/2)*(input - kernel.mean)*inv(kernel.covariance)*(input - kernel.mean)' );
```

---

boost_booster

```
function booster = boost_booster(dimX,initPopulation,noRegressors)

  booster.noRegressors = noRegressors;
  booster.initPopulation = initPopulation;
  booster.type = 'booster';
  booster.dimX = dimX;

  for i=1:noRegressors

     booster.Weights(i) = 0;
     booster.Kernel(i).mean =  zeros(1,dimX);
     booster.Kernel(i).covariance = diag(ones(1,dimX));

  end
```

---

boost tune regresor

```
function booster1 = boost_tune_regressor(training,target,booster,regressor,maxIterations,maxRepeatingTimes)

y1 = sum(target.^2);
for l=1:maxRepeatingTimes

  fprintf(' iteration = %d \n',l);
  population.weightings = ones(1,booster.initPopulation) * (1/booster.initPopulation);

% -----from min to max training-----
population.mean = max(training(:))*(2*rand(booster.dimX,booster.initPopulation) - 1);
population.covariance = abs(max(training(:)))*(rand(booster.dimX,booster.initPopulation)+0.01);

if (l > 1)
   population.mean(:,1) =  best.mean;
   population.covariance(:,1) = best.covariance;
end

population.loss  = zeros(1,booster.initPopulation);

for t=1:maxIterations
 population.best  =   [1,1];
 population.worst =   [0,1];
```

```
      for i=1:booster.initPopulation
          kernel.mean       =   population.mean(:,i)';
          kernel.covariance =   diag(population.covariance(:,i));

%-------kernel.covariance---------

         for j=1:booster.dimX
          if (kernel.covariance(j,j) < 0.01)
             kernel.covariance(j,j) = 0.01;
          end
        end

          population.loss(i) =  boost_loss(kernel,training,target,y1);

           if (population.loss(i)  < population.best(1))
             population.best = [population.loss(i) ,i];
           end
           if (population.loss(i)  > population.worst(1))
             population.worst = [population.loss(i) ,i] ;
           end
       end

   population.loss = population.loss / sum(population.loss);
   epsilon = sum(population.weightings.*population.loss);
   beta = epsilon / (1 - epsilon);

   for j=1:booster.initPopulation
     if (beta <=1)
        population.weightings(j) = population.weightings(j)*(beta^population.loss(j));
     else
        population.weightings(j) = population.weightings(j)*(beta^(1 - population.loss(j)));
     end
  end

  population.weightings = population.weightings / sum(population.weightings);


% -------  population.weightings----------
  % ----------- parameter updating -------
  new.mean        =  zeros(booster.dimX,1);
  new.covariance  =  zeros(booster.dimX,1);
  new1.mean       = zeros(booster.dimX,1);
  new1.covariance  = zeros(booster.dimX,1);

  for j=1:booster.initPopulation
       new.mean = new.mean + population.weightings(j)*population.mean(:,j);
       new.covariance = new.covariance + population.weightings(j)*population.covariance(:,j);
end

   index = population.best(2);
   mew1.mean = population.mean(:,index) +  population.mean(:,index) - new.mean;
   new1.covariance = population.covariance(:,index) +  population.covariance(:,index) - new.covariance;

   index = population.worst(2);
   Kernel(1).mean = new.mean';
   Kernel(1).covariance = diag(new.covariance);
   Kernel(2).mean = new1.mean';
   Kernel(2).covariance = diag(new1.covariance);

   for j=1:booster.dimX
    if (Kernel(1).covariance(j,j) < 0.01)
       Kernel(1).covariance(j,j) = 0.01;
    end
     if (Kernel(2).covariance(j,j)  < 0.01)
       Kernel(2).covariance(j,j) = 0.01;
    end
  end
```

```
   if (boost_loss(Kernel(1),training,target,y1) < boost_loss(Kernel(2),training,target,y1))
      population.mean(:,index) =  new.mean';
      population.covariance(:,index) =  new.covariance;
   else
              population.mean(:,index) =  new1.mean';
              population.covariance(:,index) =  new1.covariance;
   end
 end

   index = population.best(2);
   best.mean = population.mean(:,index)';
   best.covariance = population.covariance(:,index);
   end

%--- maxRepeatingTimes-------

   booster1 = booster;
   fprintf('Updating %d \n',regressor);
   booster1.Kernel(regressor).mean = population.mean(:,index)';
   booster1.Kernel(regressor).covariance = diag(population.covariance(:,index));


-----------------------------------------------------------------------------------------------------
```

boost tune

```
function booster = boost_tune(training,target,booster,maxIterations,maxRepeatingTimes,mse1)

if (booster.type ~= 'booster')
  error('Argument must be booster structure!');
end

  true_target = target;
  mse = zeros(1,booster.noRegressors);

  for i=1:booster.noRegressors
       fprintf('Tuning regressor %d \n',i);

    % --- positioning and shaping ith regressor-----
    booster = boost_tune_regressor(training,target,booster,i,maxIterations,maxRepeatingTimes);
    % --- calculating regressor weight----------------
      x = 0;
      y = 0;
      for j=1:length(target)
          x = x + target(j)*boost_output(training(j,:),booster,i);
          y = y + boost_output(training(j,:),booster,i)^2;
      end
      booster.Weights(i) = x/y;

%  - --- calculating new targets---------
    for j=1:length(target)
         target(j) = target(j) -  booster.Weights(i) * boost_output(training(j,:),booster,i);
      end

% --- calculating MSE-----------------
   input = training;
   output = zeros(1,length(input));
   mse(i) = 0;
   for k=1:length(input)
      mse(i) =  mse(i) + (true_target(k) - boost_output(input(k,:),booster))^2;
  end
    fprintf('MSE: %f \n',mse(i)/3499);
    if (i>1)
       if (mse(i))>(mse(i-1))
          error('stop');
       end
    end
         plot(mse/3499);
```

```
 end
 mse1=mse/3499
```

---

boost loss

```
function loss = boost_loss(kernel,training,target,y1)   % correlation of k-th regressor to target data
 [rows,cols] = size(training);
 loss = 0;
 g1 =  0;
 for j=1:rows
    o = boost_kernel_output(kernel,training(j,:));
    loss = loss + target(j).*o;
    g1 = g1 + o^2;
 end
      if (loss ~= 0 & g1 > 0 & y1 > 0)
          loss = 1 - abs( loss / (sqrt(g1) * sqrt(y1)) );
       else
          loss = 1;
       end
      if (isnan(loss))
          loss = 1;
       end
```

## Summary

In this thesis a novel technique is used to construct sparse generalized Gaussian Kernel regression model- so called neural network. Kernel which maximize Renyi entropy is used too. Experimental results obtained using these models are promising.

## Literatūra


1. Mechatronikos gaminių kokybė, J.Kruopis, A. Vaišvila, R.Kalnius. Vilnius, Vilniaus universiteto leidykla, 2005.
2. Sparse incremental regression modeling using correlation criterion with boosting search, S. Chen, X.X. Wang, D.J. Brown
3. Some results concerning maximum Renyi entropy distributions, O.Johnson, C.Vignat, 2006
4. A characterization of the multivariate distributions maximizing Renyi entropy, C.Vignat, 2002