Dirbtiniai neuroniniai tinklai

Š. Raudžio paskaitų konspektas

Marius Gedminas

2003 m. pavasaris
(VU MIF informatikos magistrantūros studijų 2 semestras)

Šis konspektas rinktas LaTeXu Š. Raudžio paskaitų metu. Poroje paskaitų paskaitų aš nedalyvavau ir jas paskui nusirašiau nuo kolegų (3, 4, 5, 15, 16, 17 skyriai). Deja, laiko tvarkingai viską patikrinti ir suredaguoti dar neradau, tad patariu per daug šiuo konspektu nepasitikėti. Jei rasite klaidų ar turėsite kokių pastabų, galite jas atsiųsti man elektroninio pašto adresu `mgedmin@delfi.lt`.

1 Neuroniniai tinklai

1943 m. neurono modelis:

$$s = w_0 + \sum_{i=1}^{p} w_i x_i$$

čia $x_1, \ldots, x_p$ – įėjimai. Paskui pritaikomas slenkstis ir reikšmė keičiama į 0 arba 1. Tai – neurono išėjimas.

Vėliau buvo sugalvota rezultatą perleisti pro *sigmoidinę* funkciją (angl. *logistic sigmoid*)

$$f(s) = \frac{1}{1 + e^{-s}}$$

Šios funkcijos ypatybės:

- $f(s)$ beveik tiesinė, kai $|s|$ yra mažas;
- $f(s)$ artėja prie 0, kai $s \to -\infty$;
- $f(s)$ artėja prie 1, kai $s \to \infty$.

Kartais vietoje sigmoidinės funkcijos naudojamas hiperbolinis tangentas, duodantis atsakymą iš intervalo $(-1; 1)$.

Atpažinimo uždavinys Reikia suskirstyti duomenis į klases. Tarkime, kad turime du požymius: x_1 – svoris, x_2 – ūgis. Norime atskirti berniukus nuo mergaičių. Tai gali padaryti vienas neuronas, tinkamai parinkus svorius w_i:

$$s = x_1 w_1 + x_2 w_2 + w_0$$

$s = 0$ yra skeliamasis paviršius. $s > 0$ – berniukai, $s < 0$ – mergaitės. Jei įtrauksime sigmoidinę funkciją, tuomet $f(s) > 0.5$ – berniukai, $f(s) < 0.5$ – mergaitės.

Sudėtingesniais atvejais vieno neurono nepakanka – reikia tinklo, sudaryto iš kelių sluoksnių. Pvz., keturi neuronai pirmajame sluoksnyje duoda rezultatus $y_1, \ldots, y_4$, o penktasis neuronas antrajame sluoksnyje juos ima kaip įvestis.

Dirbtinis neuroninis tinklas – rinkinys tarpusavyje sujungtų neuronų. Labiausiai paplitusios yra dvi dirbtinių neuroninių tinklų rūšys:

- *vienasluoksnis perceptronas* arba tiesiog perceptronas (angl. *single layer perceptron*; *SLP*) – tiesiog vienas neuronas.
- *daugiasluoksnis perceptronas* (angl. *multilayer perceptron*; *MLP*) – daug neuronų, išdėstytų sluoksniais. Kiekvieno sluoksnio neuronų išėjimai sujungti su kito iš eilės sluoksnio neuronų įėjimais. Įėjimo sluoksnis – pradiniai duomenys; išėjimo sluoksnis – paskutiniame sluoksnyje esantys neuronai ir jų išėjimai; visi kiti sluoksniai vadinami paslėptais.

Didžioji dauguma dirbtinių neuroninių tinklų yra SLP arba MLP.

MLP su vienu paslėtu sluoksniu gali modeliuoti bet kokio sudėtingumo atpažinimo paviršių.

Prognozavimo uždavinys Perceptronus galima naudoti ir prognozavimui.

MLP su vienu paslėtu sluoksniu gali aproksimuoti bet kokią funkciją (jei tame sluoksnyje yra pakankamai neuronų).

2 Vienasluoksniai ir daugiasluoksniai perceptronai ir jų mokymo principas

Kas yra vienasluoksniai ir daugiasluoksniai perceptronai parašyta praeitame skyriuje.

Problema: kaip reikėtų rasti perceptrono koeficientus w_i? Sprendimas: perceptroną reikia apmokyti.

Turime rinkinį mokymo vektorių $(x_{11}, \ldots, x_{1p}, t_1)$, …, $(x_{n1}, \ldots, x_{np}, t_n)$. Čia x_{ji} – įėjimo reikšmės, t_j – tikslas (*desired target*). Apibrėžkime kainos funkciją

$$c = \sum_{j=1}^{n} \left(t_j - f(w_1 x_{j1} + \cdots + w_p x_{jp} + w_0)\right)^2$$

Norime ją minimizuoti. Tai būtų trivialu, jei nebūtų netiesinės f-jos f.

Minimizavimui (apmokymui) yra daug įvairių metodų – *error back propagation*, *conjugate gradient* ir t.t.

3 Klasifikavimo uždavinys. Statistinis klasifikavimas. Mokymas vienasluoksniu perceptronu.

Kas yra klasifikavimo uždavinys – žr. pirmąjį skyrių.

Statistinis klasifikavimas Įvertiname kiekvienos klasės pasiskirstymo tankį $f_i(x)$. Tarkime, kad klasių yra dvi. Jei kažkuriame taške $f_1(x) > 0$, o $f_2(x) = 0$, aišku, jog šis taškas priklauso pirmajai klasei ir atvirkščiai, jei $f_1(x) = 0$, o $f_2(x) > 0$, taškas

priklauso antrajai klasei. Ką daryti, jei $f_1(x) > 0$ ir $f_2(x) > 0$? Galime tiesiog imti klasę, kurios tankis tame taške didesnis:

$$g(x) = \ln\left(\frac{f_1(x)}{f_2(x)}\right)$$

Jei $g(x) > 0$, reiškia, $f_1(x) > f_2(x)$ ir x priskiriame pirmajai klasei; jei $g(x) < 0$, reiškia, $f_1(x) < f_2(x)$ ir x priskiriame antrajai.

Kai kurios klasės pasitaiko dažniau nei kitos. Jei klasių pasirodymo tikimybės yra q_1 ir q_2 ($q_1 + q_2 = 1$), galime lyginti ne $f_1(x)$ ir $f_2(x)$, o $q_1 f_1(x)$ ir $q_2 f_2(x)$.

Galima taip pat atsižvelgti į klaidos kainą (geriau suklysti į saugesnę pusę).

Sprendimas vienasluoksniu perceptronu Žr. pavyzdį pirmame skyriuje.

Turime perceptroną:

$$g(x) = x_1 w_1 + x_2 w_2 + w_0$$

Kaip rasti koeficientus?

Nuostolių funkcija:

$$c = \sum_{j=1}^{n} \left(t_j - f(w_1 x_{j1} + \cdots + w_p x_{jp} + w_0)\right)^2$$

n – mokymo duomenų kiekis, p – požymių skaičius, x_j – mokymo vektorius, t_j – trokštamas išėjimas tam vektoriui.

Kas toliau?

1. *atsitiktinė paieška* – prigeneruojame 10^{20} variantų svorių ir ieškome geriausio rezultato
2. *genetiniai algoritmai* – ieškome rajono, kuriame yra teisingas variantas ir toliau dirbame tame rajone
3. *mažiausiųjų kvadratų metodas* – sprendžiame lygčių sistemą

$$\begin{cases} \frac{\partial c}{\partial w_1} = 0 \\ \frac{\partial c}{\partial w_2} = 0 \\ \vdots \end{cases}$$

Jei f-ja turi daug minimumų, gali būti sunku surasti teisingą atsakymą.

Niutono metodu

$$w_{t+1} = w_t - \eta \frac{\partial c}{\partial w}$$

η – mokymo žingsnis. Jei jis per didelis, diverguosime.

Mokymo žingsnį galima reguliuoti pagal sėkmę: jei sekasi – didiname, jei nesiseka – mažiname.

Beje, galima ir atsisakyti nuo atsakymo pateikimo, t.y. atsakyti „nežinau, modelis per silpnas šiam atvejui“.

Pradedant mokyti reikia paduoti pradinius svorius. Vienasluoksniam perceptronui jų reikšmės nesvarbios, tad galime imti nuliukus.

4 Tiesinė ir kvadratinė diskriminantinės funkcijos

Prielaida: duomenys yra normaliai pasiskirstę. $\mu x = Ex$ – vidurkis (aka matematinė viltis), $\sigma^2 = E(x-\mu)^2$ – variacija, σ – dispersija.

Normalinis tankis yra

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(x-\mu)\sigma^{-2}(x-\mu)}$$

Bendru atveju p-matėje erdvėje (daugiamatis) normalinis tankis yra

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

Čia Σ – kovariacinės matrica. x bei μ šiuo atveju yra vektoriai.

Taigi, jei turime dvi klases su vidurkiais μ_1, μ_2 ir kovariacinėmis matricomis Σ_1, Σ_2, gauname štai tokią diskriminantinę funkciją:

$$g(x) = -\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) - \frac{1}{2}ln(|\Sigma_1|) + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) + \frac{1}{2}ln(|\Sigma_2|)$$

Ji vadinama *kvadratine diskriminantine funkcija*.

Jei $g(x) > 0$, tai x priklauso pirmajai klasei, jei $g(x) < 0$, antrajai.

Dar galime pridėti kitus daugiklius (tikimybes, klaidos kainos įverčius, etc. – žr. aukščiau).

Problema: kaip turint mažai duomenų gauti kovariacinę matricą? Tada galime nutarti, kad $\Sigma_2 = \Sigma_1 = \Sigma$, nors tai netiesa. Gauname tiesinę funkciją

$$g(x) = w^T x + w_0 = \sum_i w_i x_i + w_0$$

kur $w = \Sigma(\mu_1 - \mu_2)$, $w_0 = \frac{1}{2}w^T(\mu_1 + \mu_2)$

Tai – *Fišerio tiesinė diskriminantinė funkcija*.

5 Perceptrono evoliucija mokymo metu (klasifikavimo uždavinys)

Klaidos funkcija

$$w_{sf} = \frac{1}{N}\sum_{\alpha=1}^{N}\left(t_\alpha - f(w^T x_\alpha + w_0)\right)^2$$

šią funkciją minimizuojam.

Tarkime, kad koordinates perkeliam į duomenų vidurkį, t.y. vidurkis visada yra 0.

$\Sigma\Sigma_\alpha = 0$ – vidurkis, $N_1 = N_2$ – objektų nėra(?), $w_{t=0} = 0$ – pradiniai svoriai.

$t_\alpha \in \{-1, +1\}$.

Funkcija pas mus tiesinė $f(x) = x$

$w_1 = w_0 - \frac{\partial cost}{\partial w^t} = const(\overline{x}^{(1)} - \overline{x}^{(2)})$. Geometrinė prasmė tokia: sujungiame sričių vidurkius atkarpa ir per jos vidurį išvedame statmeną tiesę.

Tai – *Euklidinio atstumo klasifikatorius*.

$D(X, \overline{X}^{(2)}) - D(X, \overline{X}^{(1)})$ – atstumai nuo vidurkio iki tiesės lygūs.

Fišerio atveju gautume $W_F = S^{-1}(\overline{X}^{(1)} - \overline{X}^{(2)})$.
Per vidurį gauname

$$W_t = \big(S + \underbrace{\frac{2}{\eta(t-1)}}_{\text{Haley narys}} \begin{pmatrix} 1 & 0 & \dots \\ 0 & 1 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}\big)^{-1}(\overline{X}^{(1)} - \overline{X}^{(2)})$$

T.y. perceptronas duoda tą patį Fišerio klasifikatorių, tik prie diagonalinių elementų pridėtas šioks toks triukšmas, kuris artėja prie nulio, kai $t \to \infty$.

Atraminių vektorių klasifikavimas Idėja: tarp artimiausių kaimynų atstumas – mažiausias. Imame tris artimiausius kaimynus (du iš vienos klasės, trečią iš kitos) ir brėžiam tarp jų tiesę, kad ji būtų labiausiai nuo jų visų nutolusi.

6 Statistinis prognozavimas

Po klasifikavimo dažniausiai sutinkamas uždavinukas yra prognozavimas. Galima prognozuoti taip:

$$y = \sum w_i x_i + w_0$$

Formulė tinka, jei turime normalinį pasiskirstymą.

Svarbiausias dalykas – atrinkti visus rodiklius x_i. Na ir žinoti, ką nori prognozuoti (y).

Vienas iš prognozavimo būdų – imti vidurkį. Kitas – imti artimiausią variantą iš turimos imties. Trečias – mažiausiųjų kvadratų metodas: minimizuojam paklaidų kvadratų sumą

$$c = \frac{1}{n}\sum_{\alpha=1}^{n}\left(y_\alpha - \sum_{i=1}^{p} w_i x_i^\alpha + w_0\right)^2$$

Taip daro statistikai. Su neuroniniais tinklais turime

$$c = \frac{1}{n}\sum_{\alpha=1}^{n}\left(y_\alpha - f\big(\sum_{i=1}^{p} w_i x_i^\alpha + w_0\big)\right)^2$$

Reikia normuoti prognoziojamą dydį, kad būtų kitimas tarp 0 ir 1:

$$c = \frac{1}{n}\sum_{\alpha=1}^{n}\left(y_\alpha - \left(f\big(\sum_{i=1}^{p} w_i x_i^\alpha + w_0\big) - 0.5\right)\cdot\beta\right)^2$$

Šitas daiktas vadinamas standartine regresija

Yra ir kitas sprendimo būdas. Kas, jei yra didelių nukrypimų?

Vidurkis skaičiuojamas

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

netipinius nukrypimus reiktų išmesti:

$$\overline{x} = \overline{x}_n \qquad \overline{x}_k = \frac{1}{\sum_{i=1}^{k} f(x_i - \overline{x}_{k-1})} \sum_{i=1}^{k} x_i f(x_i^{\alpha} - \overline{x}_{k-1})$$

kur f stengiasi numesti nukrypimus (žr paveiksliukus palme. Pvz., $f(x) = |1 - x|$ kai |x| didelis arba 1, kai x mažas). Tai yra robastinė regresija. (Robust – atsparus nukrypimams.)

Nuostolių funkcija (kurią reikia minimizuoti) viensluoksniam perceptronui norint robastinės regresijos:

$$c = \frac{1}{n} \sum_{\alpha=1}^{n} \Psi\Big(y_\alpha - \big(\sum_{i=1}^{p} w_i x_i^{\alpha} + w_0\big)\Big)$$

Kur $\Psi(x) = x^2$ kai $|x| <= z$ ir $\Psi(x) = \Psi(z)$ kai $|x| > z$. Arba galima dar užapvalinti kampus ties $|x| \approx z$.

Problema: kokio platumo z imti? Jei bus platu, bus ta pati klasikinė regresija. Jei bus siaura, ims tik labai siaurius duomenis. Menas yra nusakyti, kas yra „dideli nukrypimai“.

7 Robastiniai algoritmai klasifikacijoje

Paklaida

$$c = \sum_{\alpha} \big(t_\alpha - f\big(\sum w_i x_i^{\alpha} - w_0\big)\big)$$

(t_α – norima reikšmė (target))

Apmokymo metu svoriai laiko momentu $t + 1$ skaičiuojami

$$W(t+1) = W(t) - \eta \frac{\partial c}{\partial W}$$

(W – vektorius iš w_i. η – mokymo žingsnis (kažkaip pasirinktas))

Kadangi f-ja f užsiriečia, tai per dideli nukrypimai nuo diskriminantinės plokštumos duoda daugmaž tokią pat paklaidos vertę, kaip ir maži. Tad perceptronas automatiškai nereaguoja į tolimus nukrypimus.

Perceptronas yra automatiškai robastiškas dideliems nukrypimams. O matematikai/statistikai tik neseniai tai sugalvojo (robastinei statistikai apie 30 metų).

Beje, apie gradientinį mokymo algoritmą. Yra du režimai: stochastinis ir totalinis. Stochastinis metodas: skaičiuojam gradientą kiekvienam mokymo vektoriui atskirai ir darom pataisymą. Jei skaičiuojam gradientą visiems kartu (imam vidurkį) prieš darydami pataisymą, turime totalinį gradientą (dar kartais vadinama batch mode).

Stochastinsi lengviau išlenda iš lokalinių minimumų. Totalinis geriau konverguoja jei yra vienas gražus minimumas.

Vienasluoksniams perceptronams geriau totalinis, daugiasluoksniams geriau stochastinis.

8 Minimalios klasifikavimo klaidos ir atraminių vektorių klasifikatoriai

Prielaida: pasiskirstymas normalinis. Tada gaunam optimalų klasifikatorių. Bet jei prielaida nepatenkinta, negausim mažiausio klaidų kiekio.

Kaip padaryti tiesinę diskriminantinę funkciją, kad minimizuotume mokymo metu gautų klaidų kiekį? Literatūroje yra kokie 14-15 būdų. O perceptronai tai daro automatiškai.

Mokant perceptroną svoriai auga ir dideli nukrypimai nebeturi įtakos (suveikia funkcijos f užsirietimai). Kuo svoriai didesni, tuo tai labiau pasireiškia.

Pvz., atimam $\lambda W'W$

$$c' = \sum_{\alpha} \big(t_\alpha - f\big(\sum w_i x_i^\alpha - w_0\big)\Big)^2 - \lambda W'W$$

Tai skatina svorių augimą. (W' yra transponuotas vektorius W)

Po truputį auginant svorius po truputį mažėja klaidų skaičius. Ten faktiškai gaunasi f outputai 0 ir 1, t.y. c (variantas be $\lambda W'W$) sumuoja klaidų skaičių. Jei padalintume iš n, gautume klaidų dažnį.

Jei klaidų nėra, svoriai auga patys. Jei jie neauga, pridedam tą $-\lambda W'W$ ir priverčiam juos augti.

O dabar apie atraminių vektorių klasifikatorius.

Jei klasės yra nutolusios viena nuo kitos, tinka daug diskriminantinių plokštumų, kurios daro po 0 klaidų. Kurią reikia pasirinkti? Reikia papildomo kriterijaus. Kokio? Skonio reikalas. Vienas sprendimas: per vidurį – maksimizuoti atstumą iki artimiausių klasių atstovų. Tuos tris artimiausius vektoriukus (dvimatėj erdvėj; trimatėj bus keturi ir t.t.), nuo kurių imam tiesę, vadinam atraminiais (support vector). Kad gražiau skambėtų, turime ne klasifikatorių, o support vector machine. Beje, atraminių vektorių skaičių galima paimti pagal skonį. Galima spresti matematiškai, taikant kvadratinį programavimą. Žmonės ilgai galvojo, pirmas straipsnis apie tai buvo 1992 m. O perceptronas tai daro automatiškai, dažnai ir geriau.

Mes nagrinėsime atvejį, kai klaidų nėra (klasės nesikerta)

Manykim, kad perceptronas jau apmokytas. Kiekvienas taškas ($\equiv$ vektorius) turi indėlį į nuostolių funkciją. $f(...)$ kiekvienam taškui yra labai arti 0 arba 1. Didžiausią indėlį duoda arčiausiai plokštumos esantys vektoriai. Mokai, mokai, gauni maksimalios klaidos klasifikatorių ir jei mokai toliau, svoriai vis dar auga, plokštuma stengiasi patekti į viduriuką. Ir gauni atraminių vektorių klasifikatorių (support vector classifier arba support vector machine).

Vietoj dvimatės ervdės galima padaryti penkiamatę: x_1, x_2 turim, dar pridedam x_1^2, x_2^2, x_1x_2. Penkiamatėj erdvėj jau lengviau sudalinti taškus, kad liktų 0 klaidų (jei neišeina gauti 0 klaidų). Ir t.t. $x_1^3, x_2^3, x_1^2x_2, x_1x_2^2, ...$ Mokom iki nulinės klaidos ir tada laukiam, kol svoriai išaugs (vienas iš būdų – palaipsniui didinti žingsnį η kai pasiekiam nulinę klaidą, kitas iš būdų – įvesti tą narį $\lambda W'W$).

Apibendrinimas: yra du kriterijai. Vienas – minimizuoti mokymo klaidų kiekį. Antras – kai jų nelieka, maksimizuoti tarpą.

Perceptronas palaipsniui realizuoja visus algoritmus:

1. euklidinio atstumo

2. reguliarizuota

3. fišerio algoritmas arba
4. pseudo-inversija
5. robastinė
6. minimalios klaidos
7. atraminių vektorių klasifikatorius

9 Klasifikavimo ir prognozavimo klaidų rūšys

Prognozuoti galima viską – bet reikia žinoti, kokiu tikslumu.

Kaip vertinti tikslumą? Paskaičiuoti paklaidos vidurkį.

$$\sigma_{pr} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_{tikra,j} - y_{prognozuota,j})^2}$$

Bet jei N labai didelis (pvz., prognozuojam kažką visiems Žemės žmonėms)?

Jei turim tik dalį duomenų, tai apsimoka juos sudalinti į dvi dalis: mokymui ir testavimui. Jei viską panaudosim mokymui, gausim labai gerą prisitaikymą bet tik tiems duomenims... „Kaip meluoti su statistika."

Didėjant duomenų kiekiui (kai $N \to \infty$), $\sigma_t \to \sigma_0$, kur $\sigma_0 = \sigma_{pr}$, o σ_t – testinė klaida,

Galima apytiksliai įvertinti

$$E\sigma_t^2 = \sigma_0^2(1 + \frac{p}{N-p})$$

kur p – požymių skaičius (duomenų dimensija).

Generalizavimo klaida – testinės klaidos matematinė viltis. Kitaip tariant, laukiama testinė klaida.

Klasifikavimo klaidos yra dviejų rūšių:

1. objektą iš klasės A neteisingai priskyrėm klasei B
2. objektą iš klasės B neteisingai priskyrėm klasei A

Vienos rūšies klaidos gali kainuoti daugiau, nei kitos.

Bendra klaida

$$P = q_1 P_1 + (1 - q_1)P_2$$

kur q_1 – tikimybė, kad objektas priklauso klasei A, P_1 – pirmos rūšies klasifikavimo klaida, P_2 – antros rūšies klasifikavimo klaida.

Jei turim du duomenų gabalus su normaliniu pasiskirstymu $N(\mu_1, I)$ ir $N(\mu_2, I)$ (μ_1, μ_2 – centrai, I – vienetinė kovariacijos matrica) su atstumu δ tarp klasių (Euklidinis atstumas, t.y. $\delta = |\mu_1 - \mu_2|$), tuomet asimptotinė klasifikavimo klaida $P_\infty = \Phi(-\frac{\delta}{2})$ (kur Φ yra pasiskirstimo funkcija ar kažkas panašaus iš statistikos – $\Phi(x) = \int_0^x \phi(x)$, kur ϕ yra ta monontoniškai didėjanti nuo 0 iki 1 tankio f-ja, atrodo, $\phi(x) =$ tikimybė, kad atsitiktinis dydis yra $< x$).

Jei pasiskirstymas nėra toks gražus apvalus, paprastas atstumas nelabai veikia. Yra gudresnė Machaonobi formulė. Apibendrintas atstumas tarp klasių

$$\delta_M^2 = (\mu_1 - \mu_2)' \Sigma^{-1} (\mu_1 - \mu_2)$$

kur μ_1, μ_2 – klasių vidurkiai (vektoriai), v' – vektoriaus transponavimas, Σ – tokia baisi kovariacinė matrica (įstrižainėje dispersijos, kitur kovariacija padauginta iš kvadratinių nuokrypių sandaugos ar kžk. panašaus). Paėmę δ_M vietoje δ anoje formulėje gauname kitą klasifikavimo klaidos įvertį.

Taigi, skirtingi metodai (šiuo atveju buvo Euklido ir Fišerio) duoda skirtingas klasifikavimo klaidas.

Svarbu. Kartoju: klasifikavimo klaida priklauso nuo metodo.

Pati primityviausia prognozė (turi kažkokį paprastą pavadinimą) – kitas duomuo bus toks pats, koks šitas. Apsimoka savo prognozę palyginti su šita – jei pagerini, gerai, jei pablogini, pradėk nuo pradžių.

Dar yra koreliacijos koeficientas, kuris kažką pasako. Vidutinė kvadratinė paklaida 5 – o kas tie 5? ką tai sako? Priklauso nuo uždavinio. Fordui 51% tikslumo akcijų kurso prognozė yra priežastis padvigubinti laboratorijos finansavimą, kitur gal reikia 99% tikslumo.

NB sakoma 'klasifikavimo klaida', bet 'prognozavimo paklaida'.

10 Klasifikavimo ir prognozavimo klaidų įvertinimo būdai

Primityviausias metodas – savos imties metodas (*resubstitution*): ant tų pačių padarau, ant tų pačių testuoju. Bet jei duomenų nedaug, yra pavojus, kad prisiderinsim prie tų duomenų. Jei N ir p artimi, galim gauti labai artimą nuliui paklaidą, bet paėmus kitus duomenis gali būti nei į tvorą, nei į mietą.

$$E\sigma_R^2 \approx \frac{\sigma_0^2}{1 - \frac{p}{N-p}}$$

σ_0 – tikra paklaida, $E\sigma_R - \sigma_R$ matematinė viltis, R reiškia *resubstitution*.

Kitas būdas – testiniai duomenys (*hold-out* arba *cross-validation* metodas): apmokom su vienais duomenimis, su kitais testuojam. (Galima dar ir pakartoti kelis kartus tuos pačius duomenis skirtingai sudalinus į mokymo ir testinius.)

Kartais duomenų yra mažai ir gaila dalį aukoti tikrinimui. Tada mokom ant visų išskyrus vieną ir tikrinam ant to vieno. Paskui jį grąžinam ir išmetam kitą, mokom, ant to kito tikrinam. Ir taip su visais iš eilės. Slenkantis egzaminas (*leaving out*).

Standartinė cross-validation – dalinam į dvi dalis, ant vienos mokinam, su kita tikrinam, ir taip 2 kartus – 2-fold cross-validation. N-fold cross-validation yra tas pats, kas leaving one out. Tas tinka ir klasifikavimui, ir prognozavimui. Šis metodas tinka ir normaliniams ir nenormaliniams duomenims, bet yra jautrus duomenų homogeniškumui (mokom su Lietuvos miestų duomenim, testuojam su Mozambiko, gaunam šnipštą).

Grįžtam prie rodiklių: prognozavime tai – vidutinis kvadratinis nukrypimas bei koreliacijos koeficienta; klasifikavime – klasifikavimo klaidos įvertis $\hat{P} = \frac{n_{\text{klaidų}}}{N}$. Beje, $\hat{P}$ yra atsitiktinis dydis, pasiskirstęs pagal binominį dėsnį $B(P, N)$. Jo dispersija

$$\sigma(\hat{P}) = \sqrt{\frac{P(1-P)}{N}} \approx \sqrt{\frac{\hat{P}(1-\hat{P})}{N}}$$

(kadangi binominio skirstymo parametro P nežinome, imame $\hat{P}$ ir dėl to gauname apytiksliai). Taigi, jei žinome, kad iš 100 stebėjimų paklaida yra 10%, tai galime tikėtis realiai, kad ji bus 7%–13% (pagal vienos sigmos taisyklę), arba 3% – 16% (pagal dviejų sigmų taisyklę). Jei turime 1000 stebėjimų, tai patiklumo intervalas bus tarp 9.4% ir 10.6% (dviejų sigmų taisyklė duoda $\sim$96%).

Svarbiausia suprasti, kad įvertinimas yra atsitiktinis dydis. Kuo tiksliau norime įvertinti paklaidą, tuo didesnio N reikia (didesnis N mažina dispersiją ir galime tiksliau įvertinti paklaidą).

Viena gudrybė apmokant neuroninius tinklus: verta normalizuoti duomenis (padaryti kad sigma = 1 o vidurkis = 0). Kitas "fintas" yra juos dar ir dekoreliuoti.

11 Algoritmo sudėtingumo, mokymo duomenų kiekio ir kokybės ryšys

$$E\sigma_t^2 = \sigma_0^2(1 + \frac{p}{N-p})$$

kur p atspindi algoritmų sudėtingumą, N – duomenų kiekį, na o σ_t – kokybę.

Klasifikavimo klaidos pagal Euklido metodą (Euklidas BTW tai perceptronas po pirmos iteracijos):

$$EP_N \approx \Phi\left(-\frac{\delta}{2} \cdot \frac{1}{\sqrt{1 + \frac{2p}{N\delta^2}}}\right)$$

Jei mokom percpeptroną iki Fišerio lygio, gaunam

$$EP_N \approx \Phi\left(-\frac{\delta}{2} \cdot \frac{1}{\sqrt{1 + \frac{2p}{N\delta^2}}} \cdot \frac{1}{\sqrt{1 + \frac{p}{2N+1-p}}}\right)$$

(N – stebėjimų sk., N_1 – stebėjimų skaičius vienoje klasėje) Klaida iš pradžių didesnė, linksta smarkiau.

Kitaip tariant, kuo ilgiau mokom percpetroną (daugiau iteracijų), tuo daugiau mokymo duomenų reikia. Kuo didesnis sudėtingumas (tik čia sudėtingumas jau yra algoritmo sudėtingumas, o ne p įvertis), tuo daugiau duomenų reikia mokymui. Viena iš problemų – „permokymo“.

Kuo sudėtingesnis organizmas, tuo ilgiau mokosi. Palyginimas: katė ir studentas.

Už sudėtingesnį algoritmą moki didesniu duomenų kiekiu.

Vienas iš būdų, kaip patikrinti, ar duomenų algoritmui pakanka – naudoti slenkantį egzaminą. Su vienu paskirstymu gaunam 5% klaidų, su kitu paskirstymu gaunam 15% – blogai, didelis skirtumas, duomenų nepakanka. Reikia imti paprastesnį algoritmą.

Arba galima vertinti paklaidas teoriškai (jei duomenys normaliniai) ir paskui žiūrėti, ar praktiškai klaidų tikimybė panaši. Jei ne, blogai, nepakanka duomenų arba per sudėtingas algoritmas.

Beje, iš anksčiau: viena idėja – reikia triukšmo duomenyse. Algoritmas turi prisitaikyti prie triukšmo! Duosim idealiai švarius mokymo duomenis ir bus neatsparus tikram gyvenimui.

12 Daugiasluoksnis percpetronas ir jo mokymas

Galima taikyti atpažinimui. Pvz, angliško teksto atpažinimas: $k = 26$, po vieną klasę kiekvienai raidei. Kurioj klasėj σ_i reikšmė didžiasia, pagal tai ir atpažįstam. Trokštamas išėjimas: $\sigma_i = 1$, $\sigma_j = 0$ jei $i \neq j$.

Jei naudojam prognozavimui, dažniausiai apsieinam be sigmoidinės f-jos:

$$\sigma'_i = \sum_{j=1}^{h} v_{ij} y_j + v_{i0} kuri = 1..k$$

Arba galima normuoti į intervalą $[0, 1]$, bet praktikoje to niekas nedaro. Praktiškai visi naudoja neuroninį tinklas be netiesinių elementų išėjimo sluoksnyje.

Va tokia daugiasluoksnio perceptrono architektūra. Jo mokymas sudėtingesnis, nei vienasluoksnio perceptrono.

Daugiasluoksnis perceptronas gali daryti netiesinius atskyrimo paviršius.

Daugiasluoksnis perceptronas su vienu paslėptu sluoksniu yra universalus aproksimatorius: galima gauti bet kokio sudėtingumo atskyrimo paviršių.

Funkcijos neteisiškos, turi lokalių maksimumų, optimizavimas sudėtingas.

Kaip jį mokyti? Kaip visad, reikia įsivesti nuostolių funkciją ir ją minimizuoti.

Didžiulė nuostolių funkcijos formulė:

$$cost = \sum_{l=1}^{k} \sum_{s=1}^{k} \sum_{t=1}^{N_s} (t_{st}^l - \sigma_{st}^l)^2$$

t_{st}^l – trokštamas išėjimas ($1 \leq l \leq k$ – išėjimo neurono numeris, st – mokymo vektoriaus numeris), σ_{st}^l – gautas l-tasis išėjimas testavimo vektoriui x_{st}, k – klasių skaičius N_s – mokymo vektorių skaičius s-tajai klasei.

Išėjimas skaičiuojamas

$$\sigma_{st}^l = f(\sum_{j=1}^{h} v_{lj} y_j^{st} + v_{l0})$$

kur

$$y_j^{st} = f(\sum_{i=1}^{p} w_{ji} x_i^{st} + w_{j0})$$

paslėptų vektorių išėjimai.

Čia įėjimo vektoriai yra $\mathbf{x^{st}} = (x_i^{st})$ (s-tosios klasės t-asis vektorius, $1 \leq s \leq k$, $1 \leq t \leq N_s$, $1 \leq i \leq p$).

Mokymo algoritmo bendras principas:

$$\mathbf{W}_{z+1} = \mathbf{W}_z - \eta \frac{\partial cost}{\partial \mathbf{W}} \bigg|_z$$

kur z – mokymo žingsnis (iteracijos numeris), $\mathbf{W}$ – bendras visų perceptronų svorių vektorius.

$$\mathbf{W} = (w_{10}, w_{11}, \ldots, w_{1p}, \ldots, w_{h0}, \ldots, w_{hp}, v_{10}, v_{11}, \ldots, v_{1h}, \ldots, v_{k0}, \ldots, v_{kh})$$

Išvestinė

$$\frac{\partial cost}{\partial \mathbf{W}} = 2(t^l_{st} - \sigma^l_{st}) \cdot f'(\sum_{j=1}^{h} v_{lj} y^{st}_j + v_{l0}) \dots$$

Algoritmas vadinamas error back-propagation.

Kai svoriai maži, išvestinė didelė. Kai svoriai išauga, išvestinė priartėja prie 0, mokymas labai labai smarkiai sulėtėja.

13 Daugiasluoksnio perceptrono mokymo ypatybės

Pirma ypatybė – jis labai lėtai mokosi. Svoriams padidėjus išvestinė labai smarkiai sumažėja. Svorių labai daug. Pakliūna į lokalinius minimumus, iš jų ne visada iššoka.

Kaip su tuo kovoti? Vienas iš būdų – antros eilės metodai (skaičiuoja antros eilės išvestines, bet jie jautresni lokaliniams minimumams).

Levenberg-Market ar tai conjugate gradients metodas ima antros eilės išvestines, o kadangi jų daug ($|W|^2/2$), ima tik diagonalines. Konverguoja greičiau, bet labau greitai patenka į lokalinį minimumą.

Su lokaliniais minimumais kovoja multistart metodas. Mokai 10 kartų nuo 0 su skirtingais pradiniais svoriais. Iš jų 3 kartus gaunas gerai (3-5% klaidų), 7 labai prastai (15% klaidų)...

Kitas būdas – didinam mokymo žingsnį. Pvz, kas 50 iteracijų pažiūrim, ar sumažėjo nuostolių funkcija. Jei sumažėjo, padidinam η (padauginam iš 1.07). Jei padidėjo, sumažinam η (padauginam iš 0.7).

1986 m. Rumelharto ir dar kažkieno straipsnis pradėjo naują erą neuroniniuose tinkluose: sigmoidinė funkcija. Jie pasiūlė trokštamų išėjimų reikšmes imti 0.1 ir 0.9 vietoje 0 ir 1, kad svoriai neišaugtų per daug dideli. Tada ir išvestinė bus toli gražu ne 0.

Sumažėja plokščios vietos išėjimo sluoksnyje (paslėptame sluoksnyje nesumažėja)

Lygiai taip pat nėra ir to blogo, kas neišeitų į gerą. Išlošiam didesnę išvestinę – greitesnis mokymas. Pralošiam va ką: jei t = 0/1, tai funkcija cost yra lygi klaidų kiekui – taigi oficialiai mes minimizuojam klaidų kiekį, which gives us a warm and fuzzy feeling. Su 0.1/0.9 jau matuojam nežinia ką. Už tai mokam dar didesniu skirtumu nuo klasifikavimo klaidos.

Baisiausiais svarbus dalykas yra pradinės sąlygos. $\mathbf{W_0}$

Vienasluoksniu atveju apsimoka nustumti centrą tarp klasių į koordinačių pradžią, ir pradėti mokyti nuo nulinių svorių.

Čia neišeis: jei visus paslėptus neuronus mokysim su tai pačiais pradiniais svoriais, jie visi bus vienodi... nebebus jokio daugiasluoksnio perceptrono.

Pradinė sąlyga: $\mathbf{W}_0$ svoriai skirtingi. Paprastai juos parenka intervale $[-a, a]$. a parenkam tokį, kad pradinės įėjimo sluoksnio perceptronų reikšmės

$$\sum_{i=1}^{p} w_{ji} x^{st}_i + w_{j0}$$

reikšmės būtų pakankamai mažos.

Dažnai dar įėjimo reikšmės normalizuojamos, kad pakliūtų į intervalą $[0, 1]$ (taip daro Matlabas), arba kad jų dispersija būtų arti 1, o vidurkis 0 (taip daro Raudys).

Tai labai svarbu!

Dar apie pradines sąlygas: jei pradedi nuo gerų pradinių sąlygų ir laiku sustoji, rezultatas būna geresnis.

Kartojam:

1. kai $|W|$ auga, $f' \to 0$,
2. lokaliniai minimumai,
3. didinam arba mažinam mokymo žingsnį priklausomai nuo pasisekimo.
4. trokštami išėjimai – imam ne 0 ir 1, o 0.1 ir 0.9.
5. pralošiam lygybę cost = klaidų kiekis
6. pradinės sąlygos: svoriai skirtingi ir pakankamai maži
7. pradiniai duomenys normalizuoti – arba $0 <= x_i <= 1$, arba $Ex = 0$ (vidurkis) ir $\sigma = 1$ (dispersija)
8. gerai pradėti ir laiku sustoti!

14 Duomenų nuosavos reikšmės ir mokymo žingsnis. Duomenų transformavimas prieš mokant

Nuostolių funkcija

$$cost = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_{ij} - W' x_{ij} - W_0)^2$$

Jei $Ex = 0$, galima imti $W_0 = 0$ ir supaprastėja funkcija:

$$cost = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_{ij} - W' x_{ij})^2$$

Tarkime $\widehat{W}$ yra minimumas:

$$cost_{min} = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_{ij} - \widehat{W}' x_{ij})^2$$

tuomet

$$\begin{aligned} cost &= \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{j=1}^{N_i} (t_{ij} - (W + (\widehat{W} - W)' x_{ij})^2 \\ &= cost_{min} + (W - \widehat{W})' K (W - \widehat{W}) \\ &= cost_{min} + (W - \widehat{W})' \Phi \lambda \Phi' (W - \widehat{W}) \\ &= cost_{min} + U' \lambda U \end{aligned}$$

kur

$$K = \frac{1}{N_1 + N_2} \sum \sum x_{ij} x'_{ij} \quad \text{(kovariacinė matrica)}$$

ir

$$U = \Phi'(W - \widehat{W})$$

Kovariacinė matrica

$$K = \Phi\lambda\Phi'$$

kur Φ – ortogonali matrica ($\Phi\Phi' = \Phi'\Phi = I$ – vienetinė matrica). Matricos Φ eilutės bus matricos K tikriniai vektoriai. λ yra įstrižainė matrica, kurios įstrižainėje juos atitinkančios tikrinės reikšmės $\lambda_1, \ldots \lambda_p$ – dispersijos. Vektoriai yra kryptys, liambdos – dispersijos tomis kryptimis.

Išvestinė nepriklauso nuo $cost_{min}$ nario, tik nuo to $U'\lambda U$. Kitaip tariant, pasukam erdvę ir vietoje W gauname U. Čia lengviau konverguos. Mokymas yra

$$U_{z+1} = U_z - \eta\frac{\partial cost}{\partial U}$$

Išvestinė

$$\frac{\partial cost}{\partial U} = 2\lambda U_z$$

tad

$$U_{z+1} = (I - 2\eta\lambda)U_z$$

Kad seka konverguotų, turi būti

$$\forall i : |1 - 2\eta\lambda_i| < 1$$

(Pakanka paimti tik maksimalią λ reikšmę). Tad reikia parinkti pakankamai mažą η:

$$\eta < \frac{1}{\lambda_{max}}$$

Bet kuo mažesnė η, tuo lėtesnis mokymas.

Taigi, jei duomenys tokie bjaurūs ir nesimoko, vienas iš būdų yra pasukti erdvę ir sunormuoti duomenis:

$$Z = \lambda^{-\frac{1}{2}}\Phi X$$

Tuomet visomis kryptimis λ yra vienodi ir turime vieną bendrą η visoms kryptims. Tas labai pagreitina. Jei duomenys normaliniai, tai jau po pirmos iteracijos gauname teisingą Euklidinį klasifikatorių.

Ką daryti su daugiasluoksniu perceptronu, vienas Dievas težino – daug perceptronų, neaišku, kuria kryptimi sukti....

Prie tų punktų prirašom

9. reikia pasukti duomenis, transformacija yra $\lambda^{-1/2}\Phi$

Daugiasluoksnio perceptrono atveju duomenų nepasukiosi/nepanormalizuosi, kad lengviau būtų mokytis.

Jei skiriamasis paviršius labai jau sudėtingas ir perceptronas lėtai mokosi, tai galime pridėti triukšmo – aplink rutuliukus pribarstyti atsitiktinai daugiau rutuliukų, t.y. padidinam mokymo duomenų kiekį. Tada skiriamasis paviršius „išsitiesins“ – supaprastės (nors atsiras daugiau klaidų).

Kad svoriai neišaugtų pridedamas reguliarizavimo narys (weight decay) prie cost funkcijos:

$$... + \lambda W'W$$

priešingas efektas gaunamas (kad svoriai augtų), kai atimamas narys

$$... - \lambda V'V$$

(t.y. yra aspektų, kai šito reikia)

15 Duomenų transformacija

Prognozavimo paklaida lygi $\sigma_{pr}^2 = \sigma_0^2 \left(1 + \frac{p}{n-p}\right)$, t.y., kuo daugiau požymių imame, tuo didesnę paklaidą gauname.

Fišerio klasifikatoriaus paklaida:

$$EP_{kl} = \Phi\left(\frac{\delta}{2} \frac{1}{\sqrt{(1 + \frac{2p}{N\delta^2})(1 + \frac{p}{N_1+N_2})}}\right)$$

elgiasi taip pat.

Taigi, požymių skaičius p turi būti nedidelis. Kaip žinoti, kuriuos požymius imti?

Vienas iš būdų – *požymių išrinkimas*.

Pvz., turime požymius

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

ir atrenkame

$$X' = \begin{bmatrix} x_7 \\ x_{1012} \\ x_{38779} \end{bmatrix}$$

Kitas būdas – *transformacija (išskyrimas)*

Pvz., turime požymius

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

ir išvedame

$$Y' = \begin{bmatrix} y_1 \\ \vdots \\ y_r \end{bmatrix} = T \cdot X$$

kur $r << p$, o T – transformacijos matrica.

$$y_i = \sum_{j=1}^{p} t_{ij} x_j$$

Pagrindinių komponenčių metodas Reikia taip suprojektuoti duomenis, kad viena (pirma) kryptimi jų išsibarstymas būtų didžiausias, antra – mažesnis, trečia – dar mažesnis ir t.t. Kaip tų krypčių ieškoma: suskaičiuojam duomenims kovariacinę matricą

$$K = \frac{1}{n-1} \sum_{j=1}^{n} (X_j - \overline{X})(X_j - \overline{X})^T$$

Matricą K galima užrašyti tokiu pavidalu: $K = G \cdot D \cdot G'$, kur D yra diagonalinė matrica, o G – ortogonali matrica ($GG' = I$).

G – posūkio matrica (gaunama su minimalia paklaida).

Tai va, vietoje matricos T galima naudoti matricą G: $Y = X'G$ (X' turi vieną eilutę ir p stulpelių, iš G imame p eilučių ir tik pirmuosius r stulpelių; likusius stulpelius išmetame).

Y kovariacinė matrica yra D (tiksliau, pirmos r eilutės/stulpeliai).

Beje, šis metodas leidžia iš mažesnio skaičiaus komponenčių atstatyti likusias – t.y. turint Y galima gan tiksliai atgaminti X.

Dažnai iš pradžių požymiai išrenkami, o paskui transformuojami, t.y. naudojami abu būdai.

16 Požymių atrinkimo algoritmai

1. „Daryk, kaip darė kiti“
2. Imti požymius po vieną, su jais klasifikuoti/prognozuoti ir išrinkti tuos požymius, su kuriais geriausiai gaunasi.

   Bet šitai ne visada gerai pasiteisina: būna, kad po vieną nelabai gerai parodo, o poroj išprognozuoja klasifikatorių be klaidų,
3. Nagrinėti požymius po 2 ar daugiau. Jei požymių nedaug, tai viskas gerai, bet jei daug, tai tokių tuplų susidarys labai daug. Tada galima iš pradžių išrinkti tarkime 100 geriausių ir jau tada nagrinėti po 2 (bendru atveju po n).

   Arba galima atsitiktinai parinkti pvz. 5000 derinių po 2 ir iš jų išsirinkti geriausią.
4. Požymių atrinkimas su paskatinimu – remiantis tuo, kad geriausi požymiai dažniau pasitaiko, nei blogesni. Tai kažkaip (kaip??) atskirti m dažniausiai pasitaikančių požymių ir iš jų rinkti pilnu perrinkimu $n|$ reikalingų požymių.

   Apriorinėm tikimybėm išrenkam kažkokį požymių rinkinį. Įvertinam – pvz., paskaičiuojam klaidą. Ir taip daaaug rinkinių. Išrenkam tada dešimt geriausių rinkinių. Ir pažiūrime tuose dešimtyje, kokie požymiai geriausiai pasirodo.
5. *forward selection* – atrenkam 1 geriausią. Tada bandom jo kombinacijas su likusiais. Atrenkame antrą požymį – jau turime porą. Tada bandome tų dviejų kombinacijas su likusiais ir t.t. Šis būdas vadinamas *nuosekliu pridėjimu*. Yra dar *nuoseklus atmetimas* – analogiškai, tik atvirkščiai: atmetame blogiausius požymius. Yra dar ir šių dviejų metodų kombinacijos.
6. Genetiniai algoritmai.

Požymių rinkiniai skirtingais metodais gaunami skirtingai, bet dažniausiai jų paklaida yra panaši. Moralas: nereikia siekti tobulumo.

17 Požymių išskyrimas su neuroniniu tinklu

Klasikinis būdas išskirti požymius yra toks: turime perceptroną su $x_1, \ldots, x_p$ įėjimais, kažkiek paslėptų neuronų ir p išėjimų. Paslėptų neuronų turėtų būti r – tiek, kiek mums reikia požymių.

Idėja tokia: išėjimų reikšmės turi būti tokios pačios, kaip ir įėjimų. O vidinis paslėptas sluoksnis juos suspaudžia.

Išskirti požymiai yra vidinių paslėptų neuronų duodamos reikšmės prieš paduodant jas sigmoidinei funkcijai.

Žr. AANN – auto asociatyviniai neuroniniai tinklai

Galima pritaikyti duomenų vizualizacijai (pavaizdavimui dvimatėje ar trimatėje erdvėje).

18 Duomenų klasterizavimas (grupavimas)

Problema: nehomogeniniai duomenys, modeliavimas nesiseka. Reikia suskaidyti duomenis į klasterius ir juos modeliuoti atskirai.

Nehomogeniniai duomenys. Pasiskirstymo tankis

$$f(x) = \sum_{i=1}^{k} q_i f_i(x)$$

yra k komponenčių suma.

Reikia nehomogeninius duomenis suskirstyti į tokias klases prieš nagrinėjant.

Dažniausiai $f_i(x)$ yra normalinio pasiskirstymo – $f(x, \mu_i, \Sigma_i)$. Bet labai daug parametrų. Galime supaprastinti tardami, kad visos Σ_i yra diagonalinės matricos – netiesa, bet paprasčiau.

Daug greitesni algoritmai yra euristiniai.

Euristika – paimta iš lubų sveiko proto taisyklė pagal principą „man taip atrodo“.

Pavyzdys: n tašku. "Man atrodo", kad yra tiek grupių – taip ir suskirstom, paimam kiekvienos grupės centrą, tada tiesiog ieškom, prie kurio centro taškas artimiausias, tai grupei ir priskiriam.

Šis algoritmas vadinamas k vidurkių algoritmu (K-means algorithm).

Jis labai dažnai naudojamas. Pirmasis.

O paskui sugalvojo dar 200 ar 400. Prieš 20 metų buvo disertacijų bumas – sugalvojam naują, parodom, kad veikia, ginam disertaciją. Paskui pamatė, kad naudos 0 ir grįžo prie paprasčiausių algoritmų.

Paprasčiau paaiškinti, jei yra tik du centrai.

Imam du taškus ir vedam per vidurį statmenį. Tada vienai grupei suskaičiuojam vieną vidurkį, kitai grupei antrą. Užmirštam pirmą skiriamąją liniją ir vedam statmenį per tų dviejų vidurkių vidurį. Kartojam. Po kelių iteracijų jis nusistovi ir nustoja slankioti.

Paskui pradedi nuo pradžių su kitais pradiniais taškais – ir taip kokius 8 kartus. Paimi tą variantą, su kuriuo vidutinis atstumas iki centro yra mažiausias.

Kitas algoritmas: suskaičiuojam atstumus tarp visų mokymo vektorių porų. Artimiausias poras sujungiam poromis. Tada jungiam poras, kurių vidurkiai panašiausi. Jei kas labai toli, nejungiam. Ir t.t. Gaunam tokį medį. Tada pasirenkam kažkurį lygį ir kiek šakų yra žemiau jo, tiek ir bus klasių.

Tai vadinama *dendrograma*.

Viena neišpręsta problema – o kokią k reikšmę pasirinkti? Niekas nežino. Reikia kažkaip pasirinkti, kad išeitų paskui geriausiai.

19 Radialinių bazinių funkcijų neuroninis tinklas

Jie – antri pagal populiarumą po perceptronų. Tinka ir klasifikavimui, ir prognozavimui.

Idėja: prielaida, kad duomenys nehomogeniški. Pavyzdys: turim x, bandom prognozuoti y. Sugrupuojam į grupes su centrais c_1, ... Kiekvieną centrą c_i atitinka kažkoks y_i. Idėja daryti prognozę taip:

$$y(x) = \sum_{i=1}^{k} y_i \cdot \frac{1}{D(x, c_i)}$$

$D(x, c_i)$ – atstumas nuo x iki centro. T.y. tie, kas arčiau turi didesnę įtaką.

Iš tikrųjų funkcija yra kitokia:

$$y(x) = \sum_{i=1}^{k} y_i \cdot K(\frac{D(x, c_i)}{\lambda_i})$$

K yra varpo formos kreivė. λ įtakoja varpo platumą.

Toks būtų statistinis algoritmas. Neuroninis tinklas mokymo metu suras ir c_i ir λ_i.

Panašiai kaip ir su perceptronu: imam pradines c_i ir λ_i reikšmes (su klasterizavimu), tada imam kainos funkciją c ir gradientiniu metodu ieškom minimumo pagal visus λ_i ir c_i.

$$c = \sum (y_z - y(x_i))^2$$

Minusas: reikia daug skaičiavimo.

Galima paprasčiau: tiesiog kvantuoti pagal mokymo vektorius. T.y. jei yra arti centro, imam to centro vidurkį o nebandom skaičiuoti atstumų.

Klasifikavimas: kiekvienam klasteriui trys parametrai q_i, c_i bei λ_i:

$$y(x) = \sum_{i=1}^{k} q_i \cdot K(\frac{D(x, c_i)}{\lambda_i})$$

Pradinė q_i (i-tojo klasterio apriorinė tikimybė) reikšmė – vektorių skaičius i-tajame klaseryje padalintas iš visų vektorių skaičiaus.

Funkcija $K(s)$ paprastai yra e^{-s}.

Radialinių bazinių funkcijų neuroniniame tinkle gaunama reikšmė yra daugmaž tankio įvertinimas.

Kiek suprantu, paskui taikom skirtingiems klasteriams skirtingus jų modelius ir kombinuojam atsakymą pagal tą tankį. Bet kadangi skaičiavimų daug galima tiesiog kvantuoti ir taikyti tik vieną modelį.

Iš principo klaserių skaičiais skirtingose klasėse gali būti skirtingi. Pvz., klasifikuojam į sveikus ir sergančius. Sveiki beveik visi vienodi – vienas ar du klasteriai. Sergantys yra skirtingi – daug klasterių.

Klasteriai gali persidengti.

Kvantavime ne – griežtai nubrėšime ribas.

20 Sprendimo medžiai

Pagrindinė idėja: reikia suklasifikuoti/suprognozuoti daugiamatį vektorių. Dalinam sritį į dvi dalis, etc. Gaunam tokį medį, kurio šakose yra kažkurio požymio palyginimas

(daugiau/mažiau), o medžio lapai yra klasės. Paskui leidžiamės tuo medžiu ir žiūrim, kurioj kas pusėje. Tai ir yra sprendimo medis.

Sprendimo medis nebūtinai binarinis. Ir sprendimus galima daryti pagal daugiau nei vieną požymį.

Padarius sprendimų medį galime iš jo padaryti neuroninį tinklą.

Kaip su klasterizavimu galima pradėti inicializuoti radialinių bazinių funkcijų neuroninį tinklą, taip su sprendimų medžiu galima pradėti inicializuoti perceptroną.

Kaip tą medį parinkinėti? Pereinam visus požymius, randam slenkstį, kad kiek galima mažiau klaidų būtų.

Beje, čia irgi svarbu laiku sustoti. Per ilgai mokant bus blogai. Galima imti maksimalų klaidų skaičių lape (jei mažiau, nebeskaidom). Arba riboti medžio gylį.

Sprendimų medis naudojamas klasifikavimui vadinamas klasifikavimo medžiu.

Sprendimų medis naudojamas prognozavimui vadinamas regresijos medžiu.

Egzistuoja ir sprendimų miškai.

Šakojimo kriterijus gali būti bet koks – kai kas stato ten neuroninius tinklus...

Kaip iš sprendimų medžio gauti neuroną: Iš naujo: kiekviena sąlyga yra neuronas įėjimo sluoksnyje. Gauna reikiamus x_i ir grąžina 0 arba 1 vietoje true/false. Ten slenksčiai statūs, t.y. visi atsakymai yra 0 arba 1

Antras sluoksnis: po vieną kiekvienam lapui (atsakymui). Duoda 1 jei kiekviena šaka kelyje buvo atitinkamai 0 arba 1.

21 Genetiniai algoritmai

Vienas iš būdų, kaip optimizuoti daugiamatėje erdvėje, kur yra daug lokalinių minimumų, yra Monte-Karlo metodas (arba atsitiktinė paieška) – primėtai randomu daug taškų ir išrenki minimumą.

Modifikacija: primėtom daug taškų, randam erdvės sritį, kur geriau, tada mėtom taškus toje srityje ir t.t.

Kita modifikacija – genetiniai algoritmai. Bandoma kopijuoti gamtą. Turime (iš pradžių susigeneruojame atsitiktinai, o galima ir neatsitiktinai) seką vienetukų ir nuliukų, kurie nusako kažkokį svorių vektorių. Tiksliau, turime daug tokių rinkinių (tarkim, 1000). Apskaičiuojam kainos funkcijos reikšmę visiems. Išrikiuojam (geriausius į priekį) ir sudalinam į dvi dalis – pirmi 100 dauginsis, kiti ne. „Dauginimasis“: imam dvi sekas, sudalinam gabaliukais, imam dalį gabaliukų iš vieno, dalį iš kito. Ir t.t.

Šis algoritmas irgi gali įlįsti į lokalinį minimumą. Su tuo galima kovoti įvedam mutacijas – su tam tikra tikimybe keičiam kai kuriuos vienetukus nuliukais ir atvirkščiai.

Algoritmas labai lėtas, bet stabilus, mažiau jautrus lokaliniams minimumams.

Toks yra bazinis algoritmas, paskui galima fantazuoti.

Variantai: pirma apmokom kiekvieną kartą truputį gradientiniu metodu, o jau tada vertinam kainas ir skaičiuojam naują kartą.

Idėja: visada pasilaikyti 10 geriausių genų iš praeitos kartos.

Idėja: sudalinti į kelias grupes, jas mokyti atskirai, ir karts nuo karto suleisti tarpusavyje.

Galima dirbti ne su svoriais, o svorių skirtumu.

Galima iš pradžių padidinti mutacijų tikimybę ir sumažinti kryžminimo skaičių, o paskui, kai truputį pasimoko, mažinti mutacijų ir didinti kryžminimą. Taip galima išlošti laiko (algoritmas greičiau mokosi).

Galima lygiagrečiai vertinti skirtingas kainos funkcijas (skirtingus kriterijus). Dalis blogiausių pagal kiekvieną kriterijų žūsta.

22 Neuroninių tinklų kooperavimas

Labai svarbus klausimas. Kiekvienais metais vyksta konferencijos šia tema.

Kuo sudėtingesnis tinklas, tuo lengviau jis pakliūna į lokalinį minimumą ir tuo ilgiau mokosi, tuo daugiau resursų reikalauja. Idėja: paimti keletą neuroninių tinklų ir paskui kažkaip apjungti jų atsakymus.

Vienas, bet labai neįdomus variantas: paimti vieno tinklo atsakymą.

Kitas: imti vidurkį. Arba svorinį vidurkį. Arba balsuoti. Arba gali būti pasvertas balsavimas.

Galima skirstyti uždavinius į grupes ir apmokyti skirtingus tinklus kiekvienai grupei.

Ir taip toliau. Galima daug tokių būdų prigalvoti.

Arba galima tų tinklų atsakymus paduoti kaip įėjimus naujam tinklui.

Tokios sistemos skirtumas nuo daugiasluoksnio perceptrono yra tas, kad visi tie perceptronai apmokomi atskirai. Be to galima naudoti skirtingo tipo tinklus – pvz. radial-ir-taip-toliau, sprendimų medį etc.

Paslėptas akmuo: neuroniniai tinklai prisiderina prie mokymo duomenų ir „giriasi“, kad daro mažesnę klaidą, nei iš tiesų. Reikia tai įvertinti. Kuo sudėtingesnis tinklelis, tuo jis labiau giriasi.

Reikia duomenis skirstyti ne tik į mokymo ir testinius, bet ir į daugiau dalių ir nemokyti „boso“ su tais duomenimis, su kuriais apmokyti „pavaldiniai“.

"Boso taisyklė". Sudėtingas klausimas. Neišspręstas.

Angl. toks tinklų sujungimas yra "fusion" arba "gating rule", "combiner". Daug tų terminų yra.

Behaviour knowledge space (BKS) metodas – bandom visas neuroninių tinklų kombinacijas ir žiūrim, kokie atsakymai gaunasi.

23 Geriausio varianto parinkimo tikslumas

Neuroniniai tinklai dar taikomi ir optimizavimo uždaviniams.

Variantų yra daug. Kartais nuo pradinių mokymo sąlygų priklauso klaidų skaičius – 7 ar 17%.

Vienas iš sprendimų – bandyti kelis variantus su tais pačiais duomenimis ir parinkti geriausią.

Pvz.: apmokom 6 tinklus su mokymo duomenimis. Tikrinam testinius duomenis, gaunam skirtingas klaidas. Natūralu išsirinkti variantą su mažiausia klaida.

Bet ta klaida yra testiniams duomenims. Realiai klaida yra kitokia ir galbūt ją žinodami pasirinktumėme kitą.

Tada imam dar daugiau testinių duomenų: mokymo duomenimis apmokome, tikrinimo duomenimis paimame geriausią, testiniais duomenimis užsakovas tikrina.

Įsivaizduojama klaida – mažiausia klaida su testiniais duomenimis. Ideali klaida – mažiausia tikroji klaida (bet jos niekas nežino). Faktinė klaida – mūsų pasirinkto varianto (to, kurio testinė klaida mažiausia) tikroji klaida (kurios irgi niekas nežino).

Kuo daugiau variantų, tuo mažėja įsivaizduojama klaida. Faktinė klaida iš pradžių mažėja, o paskui pradeda augti. Kuo daugiau variantų nagrinėji, tuo labiau apsirinki.

Praktinis pasiūlymas: padalinti testinius duomenis į dvi dalis, vieną naudoti parinkimui, kitą naudoti faktinės klaidos paskaičiavimams, pasipaišyti dvi kreives (įsivaizduojamos ir faktinės klaidos bandymams). Paskui sukeisti tas dvi dalis pasipaišyti dar dvi kreives. Ir paskui pagal tai žiūrėti, kiek variantų apsimoka imti.

Tradicinis apgavystės būdas: nerodyti blogų variantų – užsakovui pateikti tik vieną, geriausią variantą.

Šiaip moralas: neturėdamas informacijos, nieko gero nepadarysi. Jei duomenų mažai, nieko nepadarysi.

A Egzamino klausimai

1. Dirbtiniai neuroniniai tinklai (DKT) klasifikavimo ir prognozavimo uždaviniuose.
2. Vienasluoksnis perceptronas (SLP) ir jo mokymo principai.
3. Tiesinė klasifikavimo taisyklė. Jos gavimas SLP pagalba. SLP mokymo algoritmas.
4. Tiesinė ir kvadratinė diskriminantinės funkcijos.
5. SLP evoliucija mokymo metu.
6. Tiesinis prognozavimas statistiniu metodu ir su SLP.
7. Robastiniai algoritmai klasifikavimo ir prognozavimo uždaviniuose.
8. Minimalios klasifikavimo klaidos ir atraminių vektorių (SVM) klasifikatoriai.
9. Klasifikavimo ir prognozavimo klaidų rūšys, tikslumo rodikliai.
10. Klasifikavimo ir prognozavimo klaidų įvertinimas.
11. Algoritmo sudėtingumo, mokymo duomenų kiekio ir gauto tikslumo ryšys.
12. Daugiasluoksnis perceptronas (DSP) ir jo mokymas.
13. DSP mokymo ypatybės.
14. Pagrindinių komponenčių ir kiti metodai duomenims vizualizuoti ir jiems transformuoti.
15. Duomenų nuosavos reikšmės ir mokymo žingsnis. Duomenų transformavimas mokymui pagreitinti.
16. Požymių atrinkimo algoritmai.
17. Daugiasluoksnio perceptrono panaudojimas informatyvių požymių išskyrimui.
18. Duomenų klasterizacija ir jos panaudojimai.
19. Radialinių bazinių funkcijų (RBF) ir mokymo vektoriaus kvantavimo (LVQ) DNT.
20. Sprendimo medžiai klasifikavimo ir prognozavimo (regresijos) uždaviniuose.
21. Genetiniai mokymo algoritmai.
22. Neuroninių tinklu kooperavimas.
23. Geriausio varianto parinkimo tikslumas.