6. Logistička regresija

Strojno učenje 1, UNIZG FER, ak. god. 2021./2022.

Jan Šnajder, predavanja, v2.3

Prošli put počeli smo razmatrati problem klasifikacije. Dali smo jedan općeniti uvod u **linearne diskriminativne modele**. Analizirali smo geometriju tih modela te naučili kako napasti višeklasni problem, dekompozicijom na niz binarnih klasifikacija. Nakon toga probali smo, ponešto oportunistički, iskoristiti linearnu regresiju za klasifikaciju. Vidjeli smo da to nije dobro završilo, jer algoritam linearne regresije nije robusan na primjere koji su daleko od granice. Identificirali smo da problem leži ili u modelu ili u funkciji kvadratnog gubitka. Ako je model linearan, onda kvadratni gubitak nije dobra funkcija gubitka za klasifikaciju.

Zatim smo opisali **perceptron**, čija je funkcija gubitka doduše oblikovana za klasifikaciju, pa nemamo problema kao s regresijom, međutim perceptron ima drugih boljki: ne konvergira kod linearno neodvojivih problema i daje različite hipoteze ovisno o početnoj inicijalizaciji težina.

Danas ćemo napokon govoriti o algoritmu koji pošteno obavlja klasifikaciju: to je **algoritam logističke regresije**. Nazivu unatoč, algoritam logističke regresije je klasifikacijski algoritam. [1] Štoviše, radi se o vrlo dobrom klasifikacijskome algoritmu koji se često koristi u praksi, i zato ga je važno znati i razumijeti.

1 Model logističke regresije

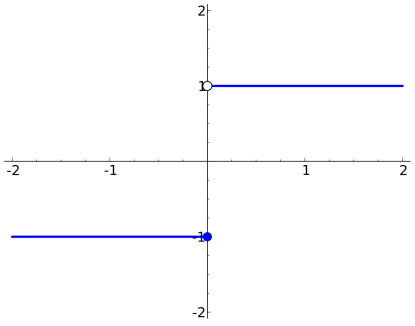
Naš pokušaj primjene regresije na klasifikaciju bio je neuspješan, ali poučan: problem je u tome što model (koji je hiperravninna) primjerima daje to veće vrijednosti što su oni dalje od granice, a funkcija gubitka to onda promptno (i drastično – kvadratno!) kažnjava.

Kako bismo to popravili, krenut ćemo najprije od toga da promijenimo model. To će nas onda automatski dovesti do drugačije funkcije gubitka (jer je gubitak povezan s modelom, budući da je gubitak definiran u odnosu na izlaz modela). Posljedično, to će nas dovesti i do drugačijeg optimizacijskog postupka (jer je optimizacijski postupak povezan s gubitkom, a koji je opet povezan s modelom). Krenimo, dakle, od modela.

Naprije, prisjetimo se modela perceptrona. Taj model definirali smo ovako:

$$h(\mathbf{x};\mathbf{w}) = f\big(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\big)$$

gdje je $f : \mathbb{R} \to \{-1, +1\}$ **funkcija praga** (step-funkcija):

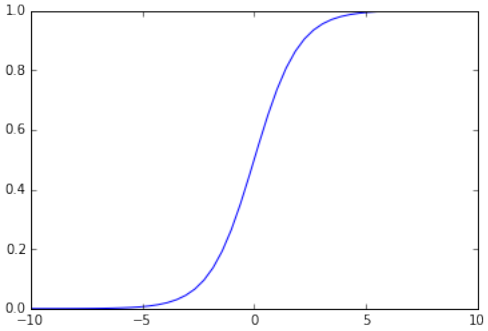


Ovakvi modeli, gdje je neka nelinearna funkcija omotana oko skalarnog umnoška vektora težina i vektora značajki, nazivaju se **poopćeni linearni modeli** (engl. *generalized linear models, GLM*). 2

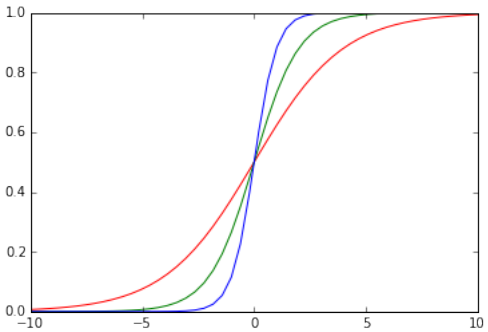
Već smo rekli da se funkcija f naziva **aktivacijska funkcija**. Tipično je to nelinearna funkcija definirana tako da vrijednost skalarnog umnoška $\mathbf{w}^{\mathrm{T}}\mathbf{x}$ preslikava na neki ograničeni interval, npr. $f : \mathbb{R} \to [0, 1]$ ili $f : \mathbb{R} \to [-1, +1]$ (to također može biti i otvoreni interval $(0, 1)$ ili $(-1, 1)$). Kako je vrijednost skalarnog produkta u intervalu od minus beskonačno do plus beskonačno, vidimo da ova funkcija zapravo "stišće" izlaz modela u ograničen interval. Zbog toga ovu funkciju ponekad zovemo i **funkcija gnječilica** (engl. *squashing function*). Koja je motivacija za to? Pa, želimo da izlaz ima ograničen raspon, kako bismo ga mogli tumačiti kao **vjerojatnost** (u slučaju intervala $[0, 1]$). Također, ne želimo da primjeri koji su jako duboko u pozitivnom ili negativnom području imaju velik izlaz modela. To je baš bio problem s linearnom regresijom! Problem je bio u tome što tamo nismo koristili nikakvu aktivacijsku funkciju. 3

Vratimo se sada na logističku regresiju. Ideja je da upotrijebimo neku aktivacijsku funkciju, ali ne funkciju praga, kao kod perceptrona. Naime, željeli bismo neku funkciju koja je slična funkciji praga, ali da je glatka, tako da je funkcija derivabilna na cijeloj svojoj domeni. Jedna funkcija koja je za to idealna je tzv. **logistička funkcija**, također poznata kao **sigmoidna funkcija** ili, za prijatelje, **sigmoida**. Ta je funkcija definirana ovako: 4

$$\sigma(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$



Nagib sigmoide može se regulirati množenjem ulaza određenim faktorom:



Zelenom krivuljom prikazan je graf fukcije $\sigma(\alpha)$, crvenom graf funkcije $\sigma(0.5\alpha)$, a plavom graf funkcije $\sigma(2\alpha)$. Vidimo da se množenjem ulaza sigmoide konstantom većom od 1 nagib sigmoide povećava. Primite ovo na znanje, jer će nam trebati malo kasnije.

Sigmoida ima **tri važne karakteristike** koje je čine dobrim odabirom za ovu našu rabotu. Prvo, vrijednosti gniječi na (otvoreni) interval $(0, 1)$. Drugo, oblikom je slična funkciji praga, što znači da će davati vrijednost blizu 1 primjerima iz jedne klase, a vrijednost blizu 0 primjerima iz druge klase. Treće, funkcija je derivabilna, što nam je važno za optimizaciju. Konkretno,

derivacija sigmoidne funkcije jest (uvjerite se u ovo!):

$$\frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}\alpha} = \frac{\mathrm{d}}{\mathrm{d}\alpha}\big(1+\exp(-\alpha)\big)^{-1} = \sigma(\alpha)\big(1-\sigma(\alpha)\big)$$

Sada možemo definirati **model** logističke regresije:

$$h(\mathbf{x};\mathbf{w}) = \sigma\big(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\big) = \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}$$

gdje smo odmah uvrstili funkciju preslikavanja iz ulaznog prostora u prostor značajki, $\boldsymbol{\phi}$, kako bismo mogli ostvariti nelinearnost granice između klasa.

Ovo je binaran klasifikacijski model. Granica između dviju klasa definirana je hiperravninom u prostoru značajki za koju vrijedi $h(\mathbf{x}) = 0.5$. U tom smislu, možemo model definirati i kao:

$$h(\mathbf{x};\mathbf{w}) = \mathbf{1}\{\sigma(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})) \geqslant 0.5\}$$

Međutim, ovakva definicija nam je manje draga. Zašto? Zato što s njom gubimo informaciju o pouzdanosti klasifikacije primjera u klasu, a to je nešto što nam sigmoidna funkcija daje. Štoviše, izlaz sigmoidne funkcije je u intervalu $(0,1)$, pa onda $h(\mathbf{x})$ možemo tumačiti kao **vjerojatnost** da primjer $\mathbf{x}$ pripada klasi s oznakom $y = 1$. Napišimo to eksplicitno: 5

$$h(\mathbf{x};\mathbf{w}) = \sigma\big(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\big) = P(y=1|\mathbf{x})$$

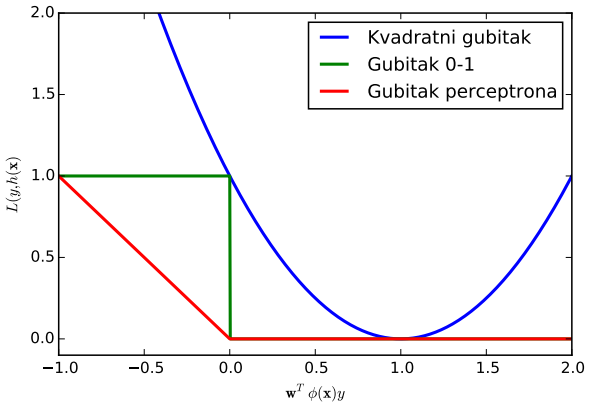
gdje je $P(y=1|\mathbf{x})$ uvjetna vjerojatnost oznake $y=1$ ako je primjer $\mathbf{x}$, tj. vjerojatnost da je oznaka primjera $\mathbf{x}$ jednaka 1.

Da sažmemo: model logističke regresije svim primjerima dodjeljuju vrijednosti iz intervala $(0,1)$, koje se dobivaju primjenom logističke funkcije na skalarni umnožak težina i značajki. Primjeri koji su daleko od granice imat će vrijednosti blizu 0 odnosno blizu 1, ovisno na kojoj su strani hiperravnine, ali nikada izvan tog intervala. Vrijednost hipoteze $h(\mathbf{x})$ možemo tumačiti kao vjerojatnost da primjer pripada pozitivnoj klasi (klasi za koju je $y=1$).

Ovime smo definirali model logističke regresije. Sljedeće što trebamo napraviti jest definirati funkciju pogreške i optimizacijski postupak. Krenimo od funkcije pogreške.

2 Pogreška unakrsne entropije

Već znamo da je funkcija pogreške očekivanje funkcije gubitka. Sjetimo se i naše skice različitih funkcija gubitaka, koju smo bili nacrtali prošli put:



Za klasifikaciju bismo željeli gubitak koji je što sličniji gubitku 0-1, ali da je derivabilan. Trebala bi nam, dakle, funkcija koja daje relativno velik gubitak za primjere koji su pogrešno klasificirani, a malen i što manji gubitak za primjere koji su ispravno klasificirani i za koje je vjerojatnost $P(y = 1|\mathbf{x})$, tj. vrijednost hipoteze $h(\mathbf{x})$, što bliže jedinici. Pored toga, želimo da je ta funkcija derivabilna. Primijetite da niti jedan od triju funkcija gubitka iz gornjeg grafa ne ispunjava sve ove uvjete.

2.1 Izvod funkcije pogreške

Mogli bismo pokušati konstruirati "ručno" takvu funkciju gubitka, ali izvlačenje funkcije gubitka iz malog prsta ne zvuči baš obećavajuće. Naime, voljeli bismo da postoji neka dublja povezanost između funkcije gubitka i modela.

Ovdje nam može pomoći da se prisjetimo kako je to bilo kod linearne regresije: pokazali smo – doduše naknadno – da su parametri koji maksimiziraju vjerojatnosti oznaka iz označenog skupa primjera isti oni parametri koji minimiziraju kvadratnu pogrešku. Tehnički, to smo izveli tako da smo najprije pretpostavili da je na oznake koje imamo u skupu označenih primjera nadodan šum koji je normalno distribuiran, gdje je srednja vrijednost te distribucije jednaka predikciji našeg linearnog modela (tj. $\mu = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$):

$$P(y|\mathbf{x}) = \mathcal{N}(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}), \sigma^2)$$

Zatim smo napisali koja je ukupna (zajednička) vjerojatnost skupa oznaka:

$$P(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)})$$

pri čemu je y oznaka jednog primjera, a $\mathbf{y}$ vektor oznaka svih primjera iz skupa $\mathcal{D}$, te smo pretpostavili da su oznake primjera nezavise i da dolaze iz iste distribucije, tj. da vrijedi pretpostavka **IID**. Vjerojatnost $P(\mathbf{y}|\mathbf{X})$ smo zatim tretirali kao funkciju parametara $\mathbf{w}$ te smo, kako bismo pojednostavili izračun, uzeli logaritam te funkcije. Na koncu smo zaključili da je negativan logaritam te funkcije proporcionalan kvadratnoj pogrešci regresije, što možemo napisati ovako: 6

$$E(\mathbf{w}|\mathcal{D}) \propto -\ln P(\mathbf{y}|\mathbf{X})$$

Iz toga je onda "prirodno" izašao kvadratni gubitak, pa smo zaključili da je **minimizacija empirijske pogreške** jednaka **maksimizaciji (logaritma) vjerojatnosti oznaka**.

Postupak koji smo upravo opisali nije bezvezan i nije slučajan. Na isti ovakav način možemo izvesti funkciju pogreške, odnosno gubitka, mnogih algoritama strojnog učenja. Ideja je, dakle, da krenemo od vjerojatnosti podataka (vektora svih oznaka $\mathbf{y}$ iz skupa $\mathcal{D}$), da napišemo tu vjerojatnost kao funkciju parametara modela $\mathbf{w}$ te da maksimiziramo logaritam te funkcije po parametrima modela. Negacija te funkcije je upravo empirijska pogreška $E(\mathbf{w}|\mathcal{D})$ koju želimo minimizirati.

Isti recept primijenit ćemo i sada: izvest ćemo funkciju pogreške logističke regresije kao negativan logaritam vjerojatnosti oznaka. Tu funkciju ćemo onda minimizirati po parametrima modela. Također, iz te će funkcije izaći funkcija gubitka, koju ćemo moći ucrtati u naš graf kao novu funkciju gubitka.

Prvi korak jest da odaberemo distribuciju kojom bismo modelirali distribuciju oznaka y za zadani primjer $\mathbf{x}$ uslijed postojanja šuma, tj. distribuciju za $P(y|\mathbf{x})$. Kod linearne smo regresije za to koristili normalnu distribuciju. To je imalo smisla, jer je kod linearne regresije oznaka y broj (numerička varijabla), pa je šum normalno distribuiran. Međutim, kod logističke regresije, budući da radimo klasifikaciju, y nije broj, nego je diskretna vrijednost (0 ili 1). U vjerojatnosnom smislu, to je onda **binarna slučajna varijabla**: varijabla koja može poprimiti vrijednost

$y = 1$ (primjer pripada klasi) ili $y = 0$ (primjer ne pripada klasi) s nekom vjerojatnošću μ odnosno $1 - \mu$. U teoriji vjerojatnosti, takva se varijabla naziva **Bernoullijeva varijabla**. Drugim riječima, kod logističke regresije oznaku y za primjer $\mathbf{x}$ modelirat ćemo Bernoullijevom distribucijom. Modeliranjem oznake kao slučajne varijable uvažavamo činjenicu da na oznaku utječe šum, tj. da je oznaka koju u skupu $\mathcal{D}$ opažamo samo jedna moguća realizacija slučajne varijable.

Prije nego što idemo u detalje, prisjetimo se što je to Bernoullijeva varijabla. Bernuoullijeva varijabla je slučajna varijabla koja ima dva moguća ishoda: događaj je ili nastupio ($y = 1$) ili nije ($y = 0$). Distribucija te varijable definirana je ovako:

$$P(y|\mu) = \begin{cases} \mu & \text{ako } y = 1 \\ 1 - \mu & \text{inače} \end{cases}$$

gdje parametar μ definira vjerojatnost nastupanja događaja, tj. vjerojatnost $P(y = 1)$. Distribucija Bernoullijeve varijable definirana izrazom s dvije grane (ako–inače) vrlo je jasna, međutim za naše potrebe nije prikladna. Naime, mi ćemo u nastavku trebati računati derivacije izraza koje sadrže vjerojatnosti Bernoullijeve varijable, a ako–inače izraze ne znamo derivirati. Ideja je onda da ako–inače izraz napišemo kao njemu ekvivalentan algebarski izraz, ali s operacijama koje znamo derivirati. To možemo lako napraviti, ovako:

$$P(y|\mu) = \mu^y (1 - \mu)^{1-y}$$

Uzmite si malo vremena da se uvjerite da je ovo isto kao i prethodni izraz, tj. da je ovo sažet način zapisa $P(1) = \mu$ i $P(0) = 1 - \mu$.

Vratimo se sada na modeliranje oznaka y. Oznake ćemo, dakle, modelirati Bernoullijevom varijablom. Sada je pitanje: odakle nam vrijednost parametra μ? Primijetimo da je, po definiciji Bernoullijeve varijable, vrijednost μ zapravo vjerojatnost da je primjer klasificiran u klasu $y = 1$. Odakle nam ta vjerojatnost? Pa, tu vjerojatnost nam daje hipoteza, tj. to je upravo izlaz modela, odnosno izlaz sigmoidne funkcije. Drugim riječima, vjerojatnost da je primjer $\mathbf{x}$ označen kao $y = 1$ je:

$$P(y = 1|\mathbf{x}) = \mu = h(\mathbf{x}; \mathbf{w})$$

Općenito, za zadani primjer $\mathbf{x}$, vjerojatnost njegove oznake (bilo $y = 1$ ili $y = 0$) prema definiciji za Bernoullijevu distribuciju jednaka je:

$$P(y|\mathbf{x}) = h(\mathbf{x}; \mathbf{w})^y \big(1 - h(\mathbf{x}; \mathbf{w})\big)^{1-y}$$

Dakle, samo smo umjesto μ uvrstili izlaz modela $h(\mathbf{x}; \mathbf{w})$, jer je izlaz modela jednak vjerojatnosti da je primjer klasificiran u klasu $y = 1$.

Sada ćemo ponoviti postupak po potpuno istom receptu koji smo koristili kod linearnog modela regresije: definirat ćemo logaritam vjerojatnosti oznaka, ovog puta uz pretpostavku Bernoullijeve distribucije za varijablu oznake y. Tu vjerojatnost tretirat ćemo kao funkciju parametra $\mathbf{w}$, i nju po tom parametru želimo **maksimizirati**. Nadalje, negativna vrijednost te funkcije jednaka je funkciji pogreške, koju onda želimo **minimizirati**. Pa, krenimo...

Ukupna (zajednička) vjerojatnost skupa oznaka je:

$$P(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)})$$

Logaritam toga je:

$$\ln P(\mathbf{y}|\mathbf{X}) = \ln \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}) = \sum_{i=1}^{N} \ln p(y^{(i)}|\mathbf{x}^{(i)})$$

Uvrstimo Bernoullijevu distribuciju, uz $\mu = h(\mathbf{x}; \mathbf{w})$:

$$\begin{aligned}\ln P(\mathbf{y}|\mathbf{X}) &= \sum_{i=1}^{N} \ln \Big(h(\mathbf{x}^{(i)}; \mathbf{w})^{y^{(i)}} \big(1 - h(\mathbf{x}^{(i)}; \mathbf{w})\big)^{1-y^{(i)}} \Big) \\ &= \sum_{i=1}^{N} \Big(y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) + (1 - y^{(i)}) \ln \big(1 - h(\mathbf{x}^{(i)}; \mathbf{w})\big) \Big)\end{aligned}$$

Emprijska pogreška jednake je negativnoj vrijednosti ove funkcije:

$$E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \Big(- y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) - (1 - y^{(i)}) \ln \big(1 - h(\mathbf{x}^{(i)}; \mathbf{w})\big) \Big)$$

Dodatno, kako ne bi ovisila o broju primjera, empirijsku pogrešku skalirat ćemo sa $1/N$:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \Big(- y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) - (1 - y^{(i)}) \ln \big(1 - h(\mathbf{x}^{(i)}; \mathbf{w})\big) \Big)$$

I to je to – izveli smo funkciju empirijske pogreške logističke regresije! Ova se funkcija pogreške naziva **pogreška unakrsne entropije** (engl. *cross-entropy error*). 8

2.2 Gubitak unakrsne entropije

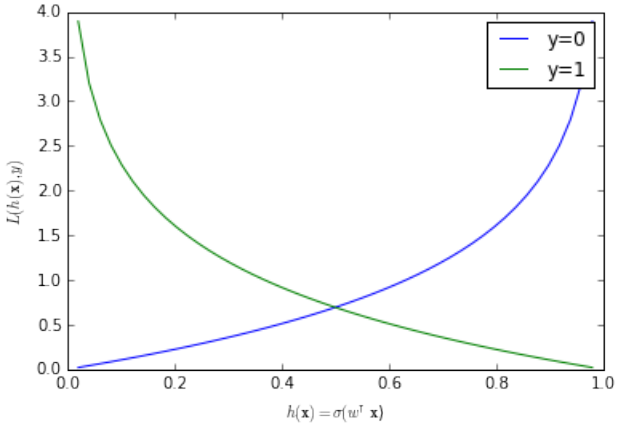
Ako je ovo što smo upravo izveli funkcija empirijske pogreške, što je onda **funkcija gubitka**? Pa, sjetimo se da je funkcija pogreške očekivanje funkcije gubitka, procijenjena kao prosjek funkcije gubitka na skupu primjera. To znači da je funkcija gubitka ono što se u izrazu za funkciju pogreške nalazi unutar izraza $\frac{1}{N}\sum_{i=1}^{N}$. Konkretno, dakle, funkcija gubitka je:

$$L(y, h(\mathbf{x})) = -y \ln h(\mathbf{x}) - (1 - y) \ln \big(1 - h(\mathbf{x})\big)$$

Ova se funkcija naziva **gubitak unakrsne entropije** (engl. *cross-entropy loss*).

Uvjerimo se da gubitak unakrsne entropije ima smisla. Izraz se sastoji od dva pribrojnika. Ako je oznaka primjera $\mathbf{x}$ pozitivna, $y = 1$, onda je drugi pribrojnik jednak nuli te samo prvi pribrojnik igra ulogu. U tom slučaju, gubitak je jednak $-\ln h(\mathbf{x})$. Ako model primjer $\mathbf{x}$ klasificira pozitivno, $h(\mathbf{x}) = 1$, onda je gubitak jednak $-\ln 1 = 0$, tj. gubitka nema. To je u redu, jer ako je $y = 1$ i $h(\mathbf{x}) = 1$, onda je klasifikacija tog pozitivnog primjera točna (istinito pozitivan primjer). S druge strane, ako model primjer $\mathbf{x}$ klasificira negativno, $h(\mathbf{x}) = 0$, onda je gubitak jednak $-\ln(0) = \infty$, tj. gubitak je maksimalan jer je pozitivan primjer klasificiran netočno (lažno negativan primjer). Slično vrijedi i za obrnutu situaciju, ako je oznaka primjera negativna, $y = 0$. Tada je prvi pribrojnik jednak nuli, a drugi pribrojnik jednak je $-\ln(1-h(\mathbf{x}))$. Ako model primjer $\mathbf{x}$ klasificira negativno, $h(\mathbf{x}) = 0$, onda je gubitak jednak nuli. S druge strane, ako model primjer $\mathbf{x}$ klasificira pozitivno, $h(\mathbf{x}) = 1$, onda je gubitak maksimalan, $-\ln(0) = \infty$. Vidimo, dakle, da je gubitak nula ako je klasifikacija točna, a da je maksimalan (beskonačan) ako je klasifikacija netočna. Sada još samo trebamo primijetiti da izlaz modela $h(\mathbf{x})$ nikada neće biti 0 niti 1, jer je kodomena sigmoidne funkcije otvoreni interval $(0, 1)$. Posljedično, gubitak za točno klasificirani primjer nikada neće biti baš nula, nego blizu nule, kao što ni gubitak za netočno klasificirani primjer neće biti beskonačno velik, nego će biti neki velik ali konačan broj. Koliko će gubitak biti blizu nuli u slučaju točne klasifikacije, odnosno koliko će biti velik u slučaju netočne klasifikacije ovisi o tome koliko je izlaz modela (sigmoidne funkcije) blizu nuli odnosno jedinici.

Ovo postaje još jasnije ako funkciju unakrsnog gubitka $L(y, h(\mathbf{x}))$ prikažemo grafički. Budući da se radi o funkciji dviju varijabli, prikazat ćemo ju s dvije krivulje, jednom za $y = 0$ i drugom za $y = 1$:

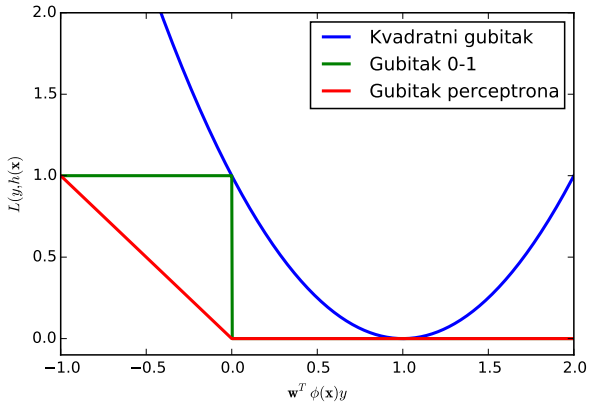


Iz grafa je opet lijepo vidljivo da gubitak to bliže nuli što je $h(\mathbf{x})$ bliži y, te da je gubitak to veći što je $h(\mathbf{x})$ dalje od y. Naravno, izlaz modela $h(\mathbf{x})$ može biti i bilo koja vrijednost između 0 i 1. Npr., koliki je gubitak na primjeru $\mathbf{x}$ za koji model daje $h(\mathbf{x}) = P(y = 1|\mathbf{x}) = 0.7$, ako je stvarna oznaka primjera $y = 0$? Iz grafa vidimo da je gubitak onda negdje između 1 i 1.5. S druge strane, ako bi izlaz modela bio isti, a oznaka primjera $y = 1$, onda bi gubitak bio manji od 0.5.

Budući da gubitka nema jedino onda kada je primjer savršeno točno klasificiran ($h(\mathbf{x}) = 1$ za $y = 1$ odnosno $h(\mathbf{x}) = 0$ za $y = 0$), a da se to ne može dogoditi jer izlaz sigmoide nikada nije 1 niti 0, zaključujemo da uvijek postoji neki gubitak. Dakle, čak i ako je primjer točno klasificiran (nalazi se na ispravnoj strani granice), ipak nanosi malen gubitak, ovisno o pouzdanosti klasifikacije. No, ono što je bitno jest da primjeri na ispravnoj strani granice ($h(\mathbf{x}) \geqslant 0.5$ za $y = 1$ odnosno $h(\mathbf{x}) < 0.5$ za $y = 0$) nanose puno manji gubitak od primjera na pogrešnoj strani granice.

2.3 Usporedba funkcija gubitaka

Sada kada smo izveli funkciju gubitka unakrsne entropije, vratimo se na grafikon funkcija gubitaka i usporedimo tu funkciju s drugim funkcijama gubitka s kojima smo se već upoznali. Prisjetimo se, taj grafikon izgleda ovako:



Međutim, da bismo to mogli napraviti, funkciju gubitka unakrsne entropije, koja je funkcija dvije varijable, trebamo preformulirati tako da to bude funkciju jedne varijable. Konkretno, trebamo napraviti isto što smo napravili i za funkciju kvadratnog gubitka: prebaciti se sa skupa oznaka $\{0, 1\}$ na skup oznaka $\{-1, +1\}$ te zatim funkciju gubitka preformulirati u funkciju od $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y$. To možemo lako napraviti, no ovdje ćemo to preskočiti i umjesto toga odmah dati konačan rezultat:

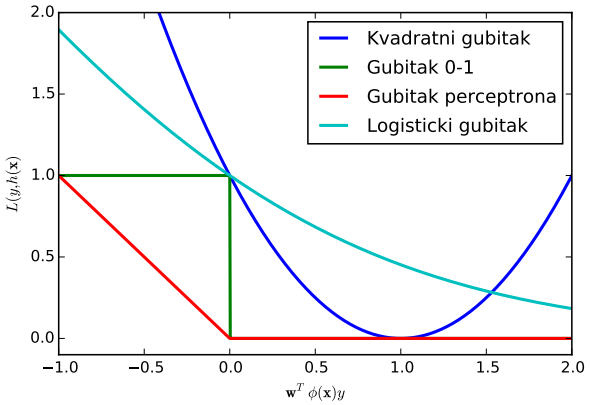
$$L\big(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y\big) = \ln\big(1 + \exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y)\big)$$

9

Ovo se sada zove **funkcija logističkog gubitka** (engl. *logistic loss*). Mala nezgodancija s ovom funkcijom jest da, za primjer koji se nalazi točno na granici, $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = 0$, vrijednost funkcije je $\ln(1+\exp(0)) = \ln 2$. To nam ne odgovara za naš grafikon, u kojemu sve funkcije gubitka (osim za gubitak perceptrona) vrijedi $L(0) = 1$. Zato ćemo napravili malu korekciju, ovako: 10

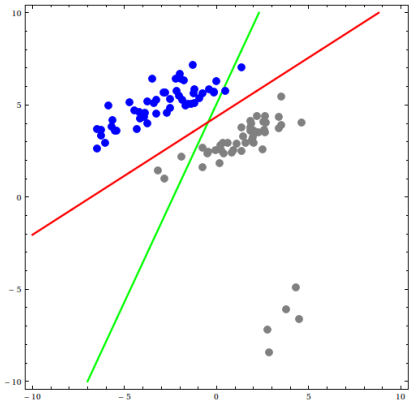
$$L\big(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y\big) = \frac{1}{\ln 2}\ln\big(1+\exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y)\big)$$

Ovime smo samo malo (za konstantan faktor $\frac{1}{\ln 2}$) skalirali vrijednost funkcije gubitka, kako bi vrijedilo $L(0) = 1$. Time se u suštini ništa ne mijenja, samo što graf izgleda urednije. I sada, napokon, funkciju logističkog gubitka možemo ucrtati u grafikon funkcija gubitaka:



Vidimo da funkcija logističkog gubitka kažnjava netočno klasificirane primjere (negativan dio x-osi) iznosom većim od jedan, i da taj iznos raste što je primjer manje točno klasificiran (tj. što je dalje od granice na njezinoj pogrešnoj strani). S druge strane, gubitak za točno klasificirane primjere (pozitivan dio x-osi) je manji od onih za netočno klasificirane primjere, te pada što je primjer točnije klasificiran (tj. što je dalje od granice na ispravnoj strani). Međutim, primijetimo da gubitak nikada nije nula – čak i točno klasificirani primjeri nanose gubitak. Kao što smo već rekli, to je zato što izlaz modela $h(\mathbf{x})$ nikada nije točno 0 niti 1. Isto možemo zaključiti iz definicije funkcije logističkog gubitka kao funkcije jedne varijable: što je točno klasificirani primjer dalje od granice, to je $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y$ većeg iznosa, i to je $\ln(1+\exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y))$ bliži nuli, međutim nikada nije jednak nuli.

Sve ovo znači da, za razliku od linearne regresije, logistička regresija neće imati problem s primjerima koji su daleko od granice, a na njezinoj ispravnoj strani. Ilustrirajmo to na binarnoj klasifikaciji u dvodimenzijskome ulaznom prostoru:



Ovaj smo primjer već bili razmotrili prošli put. Zelenom linijom prikazana je granica između dviju klasa (plavi i sivi primjeri) koju dobivamo linearnom regresijom treniranom tako da primjerima jedne klase pridjeljuje vrijednost 1 a drugima vrijednost 0 (ili -1). Problem je što

kod kvadratnog gubitka primjeri koji su daleko od granice (grupa sivih primjera dolje desno) nanose velik gubitak, neovisno o tome jesu li točno ili netočno klasificirani. Zbog toga dobivamo hipotezu koja netočno klasificira neke od primjera iz skupa označenih primjera, premda je taj skup linearno odvojiv. Crvenom linijom prikazana je granica između klasa koju dobivamo algoritmom logističke regresije. Primjeri koji su daleko od granice a točno klasificirani (grupa sivih primjera dolje desno) ne nanose velik gubitak (kao što smo vidjeli, gubitak teži k nuli što je točno klasificirani primjer pozicioniran dalje od granice), pa stoga nemaju velik utjecaj na položaj granice. Naravno, kad bi ti primjeri bili primjeri druge klase (plavi), onda bi oni nanosili značajan gubitak, budući da bi se u tom slučaju ti netočno klasificirani primjeri nalazili daleko od granice i to na njezinoj pogrešnoj strani.

2.4 Sažetak

Sažmimo do sada učinjeno. Model logističke regresije definirali smo ovako:

$$h(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})) = \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))} = P(y=1|\mathbf{x})$$

zadnji izraz nas treba podsjetiti da je izlaz logističke regresije zapravo jednak vjerojatnosti da je primjer označen pozitivno. Zato kažemo da je logistička regresija **probabilistički klasifikator**. Logistička regresija je ujedno **diskriminativan klasifikacijski model**, jer modelira isključivo granicu između klasa (pomoću težina $\mathbf{w}$), a ne i distribuciju klasa, kao što bi to radili generativni modeli.

Funkciju pogreške zatim smo izveli kao negativan logaritam vjerojatnosti oznaka iz skupa $\mathcal{D}$, ovako:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\Big(-y^{(i)}\ln h(\mathbf{x}^{(i)};\mathbf{w}) - (1-y^{(i)})\ln\big(1-h(\mathbf{x}^{(i)};\mathbf{w})\big)\Big)$$

iz čega smo iščitali funkciju gubitka unakrsne entropije:

$$L(y, h(\mathbf{x};\mathbf{w})) = -y\ln h(\mathbf{x};\mathbf{w}) - (1-y)\ln\big(1-h(\mathbf{x};\mathbf{w})\big)$$

Ovdje želim da primijetite da smo funkciju gubitka izveli iz funkcije pogreške, a ne obrnuto. To možda izgleda neobično: znamo da je funkcija pogreške očekivanje funkcije gubitka, pa se možda čini smislenijim da prvo definiramo funkciju gubitka, a onda funkciju pogreške jednostavno definiramo kao njezino očekivanje. Međutim, oba su pristupa jednako legitimna: budući da su pogreška i gubitak povezani, možemo krenuti od pogreške i izvesti gubitak, ili obrnuto. Ovisno o vrsti algoritama, nekad je smisleniji jedan a nekad drugi pristup. Kod probabilističkih modela (modela koji modeliraju vjerojatnost da primjer pripada klasi), kao što je to slučaj s logističkom regresijom, smislenije je krenuti od funkcije pogreške, odnosno negativnog logaritma vjerojatnosti oznaka, a onda iz toga iščitati funkciju gubitka. 11

3 Gradijentni spust

Definirali smo dvije komponente algoritma logističke regresije: model i funkciju gubitka (i pripadnu funkciju pogreške). Nedostaje nam još samo treća komponenta, a to je postupak optimizacije. Kao i uvijek, postupak optimizacije treba nam dati težine $\mathbf{w}$ koje minimiziraju empirijsku pogrešku na skupu za učenje:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} E(\mathbf{w}|\mathcal{D})$$

Kao što smo radili do sada, možemo pokušati izraziti gradijent za $E(\mathbf{w}|\mathcal{D})$ po $\mathbf{w}$, izjednačiti to s nulom i tako analitički pronaći minimizator $\mathbf{w}^*$. Nažalost, kod logističke regresije to neće

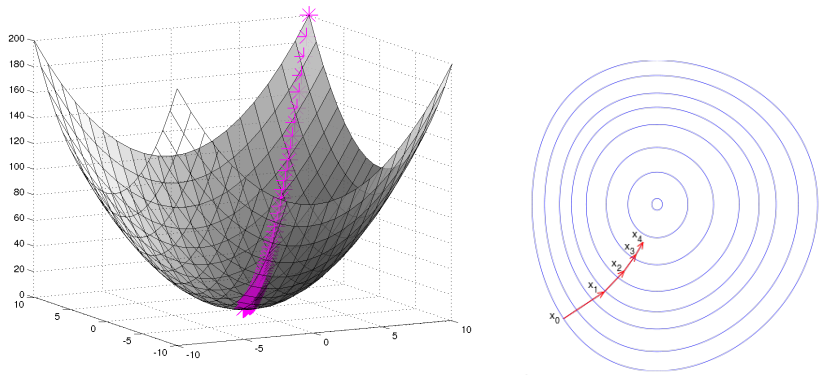
funkcionirati jer u ovom slučaju izlaz modela nelinearno ovisi o težinama $\mathbf{w}$ (zbog nelinearne aktivacijske funkcije σ). Zbog toga za ovaj optimizacijski problem nažalost **ne postoji rješenje u zatvorenoj formi**! 12

Sličan problem imali smo i kod perceptrona. Kako smo ga tamo riješili? Radili smo **gradijentni spust**. Kod perceptrona je dodatan problem bio taj što funkcija praga nije neprekidna, ali to smo riješili tako da smo derivirali samo za primjere koji su netočno klasificirani (za primjere koji su točno klasificirani ionako je derivacija bila jednaka nuli).

I kod logističke regresije ćemo također raditi gradijentni spust. To nije jedina mogućnost, ali je najjednostavnija. Prisjetimo se, ideja gradijentnog spusta jest da se, krenuvši od neke početne točke (početnog vektora $\mathbf{w}$), postepeno "spuštamo" niz površinu funkcije $E(\mathbf{w}|\mathcal{D})$ u smjeru suprotnome od gradijenta u točki $\mathbf{w}$. To ponavljamo dok se ne spustimo u točku u kojoj je gradijent jednak nuli ili je barem dovoljno blizu nuli. Formalno, težine $\mathbf{w}$ u svakoj iteraciji gradijentnog spusta ažuriramo na ovaj način: 13

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}|\mathcal{D})$$

gdje je η **stopa učenja** koja određuje veličinu koraka koje radimo spuštajući se prema minimumu. Ako radimo korake u dotičnom smjeru, spustit ćemo se do minimuma. Npr., u dvodimenzijskom prostoru parametara to bi moglo izgledati ovako:

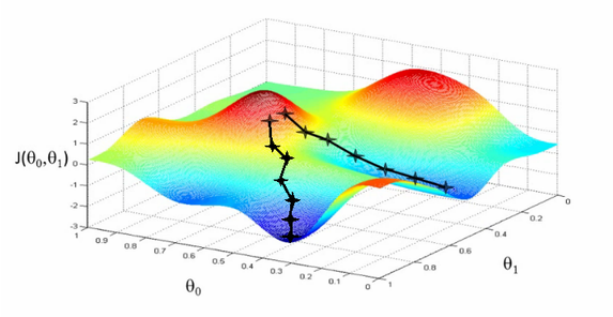


Lijeva slika prikazuje površinu funkcije pogreške $E(\mathbf{w}|\mathcal{D})$ niz koju se gradijentnim spustom iterativno spuštamo do minimuma (ljubičasti trag). Na desnoj su slici prikazane izokonture te funkcije pogreške i crvenim je vektorima naznačena staza kojom se gradijentim spustom spuštamo do minimuma funkcije (u sredini izokontura). Vektori odgovaraju negativnom gradijentu u točki funkcije pogreške u ishodištu vektora. Vektori su uvijek okomiti na izokonturu u točki ishodišta vektora, jer je smjer koji je okomit na izokonturu smjer najbržeg smanjivanja vrijednosti funkcije. Također možemo vidjeti da su vektori to manji što smo se spustili bliže minimumu, jer je nagib površine funkcije pogreške sve manji kako se primičemo točki minimuma.

Da bismo, dakle, algoritmom gradijentnog spusta pronašli minimum funkcije empirijske pogreške logističke regresije, trebat ćemo izračunati gradijent te funkcije. No, prije nego što to učinimo, pogledajmo nekoliko detalja.

3.1 Konveksnost

Gradijentni spust dovest će nas to točke za koju vrijedi $\nabla E(\mathbf{x}; \mathbf{w}) = 0$. Je li ta točka nužno minimum funkcije pogreške? Minimum da, ali to ne mora nužno biti globalni minimum. Naime, funkcija pogreške općenito može imati jedan ili više **lokalnih minimuma**. U tom slučaju gradijentni spust, ovisno o početnoj točki, lako može zaglaviti u nekom od lokalnih minimuma. Na primjer:



Slika prikazuje funkciju pogreške koje ima nekoliko lokalnih minimuma. Ovisno o početnoj točki iz koje kreće gradijentni spust, možemo završiti u nekom od tih lokalnih minimuma.

Iz navedenog razloga u strojnom učenju posebno volimo funkcije koje su **konveksne**. Takve funkcije imaju samo jedan minimum, dakle, taj jedan minimum onda je sigurno i globalni minimum jer lokalnih minimuma nema. Optimizacijom konveksnih funkcija bavi se **konveksna optimizacija**, koja je jako važna za strojno učenje (možete o njoj razmišljati kao o starijoj sestri strojnog učenja; druga starija sestra je statistika). 14 15

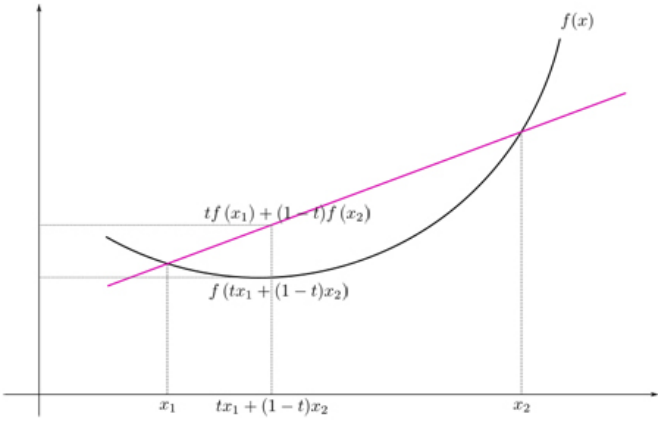
Neformalno, konveksna funkcija je ona koja je oblika "zdjele". Malo formalnije, funkcija $f : \mathbb{R}^n \to \mathbb{R}$ je **konveksna** akko: (1) domena of f, $\text{dom}(f)$, je konveksan skup i (2) svaka točka na svakoj sekanti funkcije (spojnici između dvije točke na krivulji) uvijek je veća ili jednaka od vrijednosti funkcije. Još formalnije, funkcija f je konveksna ako i samo ako:

1. Njezina domena $\text{dom}(f)$ je **konveksan skup**:

$$\forall \mathbf{x}_1, \ldots, \mathbf{x}_n \in \text{dom}(f), \forall \alpha_1, \ldots \alpha_n . \sum_i \alpha_1 = 1 \quad \text{vrijedi} \quad \sum_{i=1}^{n} \alpha_i \mathbf{x}_i \in \text{dom}(f)$$

2. Za svaki $\mathbf{x}_1, \mathbf{x}_2 \in \text{dom}(f)$ i svaki $t \in [0, 1]$ vrijedi:

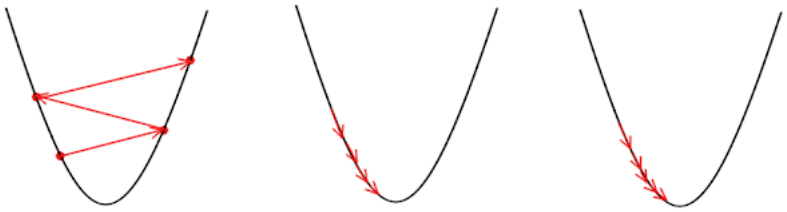
$$f(\mathbf{x}) = f\big(t\mathbf{x}_1 + (1-t)\mathbf{x}_2\big) \leqslant tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2)$$



Kada će funkcija empirijske pogreške biti konveksna funkcija? Pa, prvo trebamo primijetiti da je empirijska pogreška definirana kao zbroj gubitaka. Zatim trebamo znati da je svojstvo konveksnosti aditivno: zbroj konveksnih funkcija je također konveksna funkcija. Dakle, konkveksnost funkcije pogreške će ovisiti o funkciji gubitka: funkcija pogreške bit će konveksna ako i samo ako je funkcija gubitka konveksna. Mnoge funkcije pogreške, koje su nama zanimljive u strojnom učenju, jesu konveksne funkcije. Npr., funkcija pogreška regresije (suma reziduala) je konveksna funkcija. Također, pogreška unakrsne entropije je konveksna funkcija, budući da je i gubitak unakrsne entropije konveksna funkcija, u što se možete uvjeriti ako pogledate graf te funkcije. 16

3.2 Stopa učenja i globalna konvergencija

Rekli smo da konstanta η određuje **stopu učenja**, odnosno koliko velike korake radimo dok se spuštamo. Stopa učenja treba biti odabrana tako da nije ni prevelika ni premalena. Naime, ako odaberemo preveliku stopu učenja, može nam se dogoditi da postupak ne konvergira već da **divergira**. S druge strane, ako odaberemo premalenu stopu učenja, postupak će doduše konvergirati, ali će konvergencija biti spora:



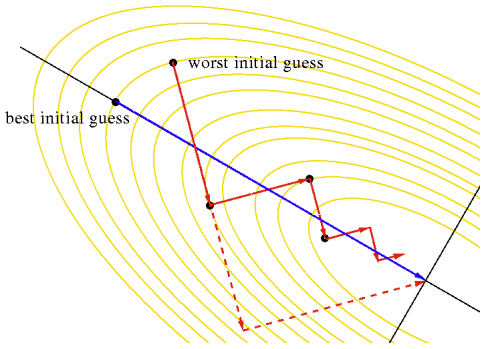
Na lijevoj slici prikazana je situacija kada je stopa učenja prevelika, pa postupak divergira. Na desnoj slici prikazana je obrnuta situacija: kada je stopa učenja premala, pa postupak sporo konvergira. U sredini je prikazana situacija za optimalnu stopu učenja.

Kako odrediti optimalnu stopu učenja? Za početak, željeli bismo imati jamstvo da će gradijentni spust konvergirati, neovisno o tome odakle započinje spust. To se svojstvo naziva **globalna konvergencija**. Divergenciju možemo spriječiti – i time ostvariti globalnu konvergenciju – tako da u svakoj iteraciji spusta odaberemo optimalnu stopu učenja. Ideja je da, nakon što izračunamo gradijent, pretražujemo u smjeru suprotnome od smjera gradijenta sve dok ne dođemo do mjesta minimuma funkcije u tom smjeru, i da napravimo korak točno do tog mjesta. Ta se tehnika naziva **linijsko pretraživanje** (engl. *line search*). Dakle, kod linijskog pretraživanja, ako je $\Delta\mathbf{x} = -\nabla f(\mathbf{x})$ smjer spuštanja, onda optimiramo funkciju: 17

$$g(\eta) = f(\mathbf{x} + \eta\Delta\mathbf{x})$$

odnosno uzimamo η koji minimizira g. Primijetite da se linijsko pretraživanje svodi na optimizaciju, odnosno nalaženje minimuma, funkcije jedne varijable (dakle u jednoj dimenziji, tj. duž pravca). Za pretraživanje duž pravca tipično se koristi pretraživanje koje (velik) početni interval rekurzivno dijeli na podintervale ili se koristi algoritam linijskog pretraživanja s povratkom. 18

Pogledajmo primjer linijskog pretraživanja u dvodimenzijskome prostoru parametara: 19



Slika prikazuje izokonture funkcije pogreške u prostoru parametara. Plavom je linijom označen trag gradijentnog spusta u optimalnom slučaju, kada je gradijent izračunat u početnoj točki direktno vodi do minimuma funkcije. Crvenom je označen trag spusta za najgori slučaj, kada je gradijent u početnoj točki pod kutem od 45° u odnosu na gradijent optimalnog slučaja. Od početne točke, linijskim pretraživanjem došli bismo do točke koja je minimum duž smjera spusta, a to je točka u kojoj je smjer spusta tangencijalan na izokonturu (na slici to je crna točka neposredno prije iscrtkane crvene linije). U toj točki ponovo računamo gradijent, koji će

biti okomit na izokonturu, te nastavljamo s linijskim pretraživanjem u smjeru suprotnome od gradijenta, itd.

Alternativa linijskom pretraživanju jest da koristimo neku od varijanti gradijentnog spusta s **adaptivnom (prilagodivom) stopom učenja**. Mi se tim postupcima nećemo baviti. 20

3.3 Stohastički gradijentni spust i online učenje

Znamo već da je funkcija pogreške očekivanje funkcije gubitka, a da to očekivanje aproksimiramo prosječnom vrijednošću funkcije gubitka na skupu za učenje. Prema tome, funkcija pogreške zapravo je zbroj funkcija gubitka za svaki primjer (do na faktor $\frac{1}{N}$): 21

$$E(\mathbf{w}|\mathcal{D}) \propto \sum_{i=1}^{N} L\big(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w})\big)$$

To nam onda daje dvije mogućnosti kako ažurirati težine kroz iteracije gradijentnog spusta: ili ćemo (1) napraviti derivaciju gubitaka za sve primjere iz skupa za učenje i onda zakoraknuti prema minimumu ili ćemo (2) za svaki primjer pojedinačno napraviti korak prema minumumu. Prvi pristup nazivamo **standardni gradijentni spust** ili **grupni gradijentni spust** (engl. *batch gradient descent*). Drugi pristup nazivamo **stohastički gradijentni spust** (engl. *stochastic gradient descent,* **SGD**), i kod njega težine ažuriramo za svaki primjer zasebno. Formalno, kod standardnog gradijentnog spusta težine ažuriramo na sljedeći način:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{i=1}^{N} \nabla_{\mathbf{w}} L\big(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w})\big)$$

dok kod stohastičkog gradijentnog spusta to radimo ovako:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L\big(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w})\big)$$

Primijetite da smo u oba slučaja zanemarili konstantu $\frac{1}{N}$. Ona se može apsorbirati u stopu učenja η (koja onda, doduše, ovisi o broju primjera N). 22

Stohastički gradijentni spust ima neke prednosti pred standardnim gradijentnim spustom. Npr., stohastički gradijentni spust manje je računalno zahtjevan od standardnog gradijentnog spusta, koji, prije nego što napravi korak, mora pozbrajati gradijent za svaki primjer. Razlika pogotovo dolazi do izražaja kod velikih skupova podataka (engl. *big data*), a problem u tom slučaju može biti i da sve primjere ne možemo učitati u memoriju. Stohastičkim gradijentnim spustom možemo u istom vremenu napraviti više koraka, što često znači da možemo doseći točku bližu globalnom optimumu. Nadalje, stohastički gradijentni spust u pravilu radi bolje kada funkcije pogreške nije konveksna, već ima mnoge lokalne minimume. Naime, u tom slučaju stohastičko krivudanje pri spustu može pomoći da algoritam ne zaglavi u lokalnom optimumu. 23

Kompromisno rješenje između standardnog i stohastičkog gradijentnog spusta, a koje se pokazalo da u praksi radi najbolje, jest da se gradijent izračunava za manje, slučajno uzorkovane grupe primjera, tzv. **minigrupe** (engl. *minibatches*). Gradijentni spust s minigrupama u pravilu radi bolje od standardnog i stohastičkog gradijentnog spusta s pojedinačnim primjerima. To je zato jer uvodi dovoljno stohastičnosti da pretraga ne zapinje tako lako u lokalnom optimumu, ali je opet dovoljno usmjeren da pretraga ne krivuda previše i time završi u suboptimalnoj točki.

24

U praksi, implementacije algoritma logističke regresije tipično koriste stohastički gradijentni spust. Motivacija za to nije izbjegavanje lokalnih optimuma (njih nema jer je funkcija pogreške unakrsne entropije konveksna), već brzina konvergencije i stabilnost rješenja. Napomenimo još 25
da se linijsko pretraživanje u tom slučaju ne koristi. Naime, kod stohastičnog gradijentnog spusta (također i kod gradijentnog spusta s minigrupama) smjer spusta ionako je aproksimacija pravog gradijenta u toj točki, pa nema smisla trošiti računalno vrijeme na odabir optimalne stope učenja. 26

4 Gradijentni spust za logističku regresiju

Sada kada sve ovo znamo o gradijentnom spustu, možemo izvesti gradijentni spust specifično za logističku regresiju, tj. za pogrešku unakrsne entropije. Prisjetimo se, pogreška unakrsne entropije je:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\Big(-y^{(i)}\ln h(\mathbf{x}^{(i)};\mathbf{w}) - (1-y^{(i)})\ln\big(1-h(\mathbf{x}^{(i)};\mathbf{w})\big)\Big)$$

Znamo da je funkcija pogreške prosjek funkcija gubitka po svim primjerima, pa je onda gradijent funkcije pogreške jednostavno prosjek gradijenata funkcije gubitka:

$$\nabla_{\mathbf{w}} E(\mathbf{w}|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{w}} L\big(y^{(i)}, h(\mathbf{x}^{(i)};\mathbf{w})\big)$$

Za stohastički gradijentni spust trebat će nam samo ∇L, dok ćemo za standardni (grupni) gradijentni spust ukupni gradijent dobiti zbrajanjem gradijenata za pojedinačne primjere, prema gornjoj formuli. Izračunajmo onda najprije gradijent funkcije gubitka. Funkcija gubitka je:

$$L(y, h(\mathbf{x})) = -y\ln h(\mathbf{x}) - (1-y)\ln\big(1-h(\mathbf{x})\big)$$

Gradijent te funkcije po težinama $\mathbf{w}$ je:

$$\begin{aligned}\nabla_{\mathbf{w}} L\big(y, h(\mathbf{x})\big) &= \Big(-\frac{y}{h(\mathbf{x})} + \frac{1-y}{1-h(\mathbf{x})}\Big)h(\mathbf{x})\big(1-h(\mathbf{x})\big)\boldsymbol{\phi}(\mathbf{x}) \\ &= \big(h(\mathbf{x})-y\big)\boldsymbol{\phi}(\mathbf{x})\end{aligned}$$

(Uvjerite se da možete sami napraviti ovaj izvod.) Ovdje smo, budući da je funkcija L kompozicija nekoliko funkcija, za deriviranje primijenili pravilo ulančavanja te iskoristili već ranije spomenutu činjenicu da je derivacija sigmoidne funkcije (a ovdje je to funkcija $h(\mathbf{x};\mathbf{w})$) jednaka $\sigma(\alpha)\big(1-\sigma(\alpha)\big)$. Taj se izraz, međutim, pokratio, pa smo u konačnici dobili prilično elegantan izraz za gradijent funkcije gubitka. Vidimo da je gradijent gubitka za primjer $\mathbf{x}$ zapravo jednak vektoru $\phi(\mathbf{x})$ (vektor primjera $\mathbf{x}$ u prostoru značajki) pomnoženom sa skalarom $h(\mathbf{x})-y$, što je razlika predikcije modela i točne oznake za primjer. To intuitivno ima smisla, jer što se predikcija modela $h(\mathbf{x})$ i točna oznaka y za neki primjer više razlikuju, to je veći nagib funkcije gubitka (pogledajte graf funkcije gubitka unakrsne entropije), odnosno to je veći utjecaj promjene težina na gubitak za dotični primjer.

Ovo nam je već dovoljno za stohastički gradijentni spust. Ako želimo raditi standardni (grupni) gradijentni spust, onda gradijent funkcije pogreške po težinama $\mathbf{w}$ izračunavamo kao:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{i=1}^{N}\big(h(\mathbf{x}^{(i)}) - y^{(i)}\big)\boldsymbol{\phi}(\mathbf{x}^{(i)})$$

gdje smo faktor $1/N$ ispustili, jer ga se može apsorbirati u stopu učenja η.

Napokon, sve ovo sada možemo složiti u algoritam optimizacije težina logističke regresije gradijentnim spustom:

▸ **Logistička regresija – standardni (grupni) gradijentni spust**

$\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
ponavljaj do konvergencije
  $\Delta\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
  za $i = 1, \ldots, N$
    $h \leftarrow \sigma(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}^{(i)}))$
    $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} - (h - y^{(i)})\,\boldsymbol{\phi}(\mathbf{x}^{(i)})$
  $\eta \leftarrow$ optimum linijskim pretraživanjem u smjeru $\Delta\mathbf{w}$
  $\mathbf{w} \leftarrow \mathbf{w} + \eta\Delta\mathbf{w}$

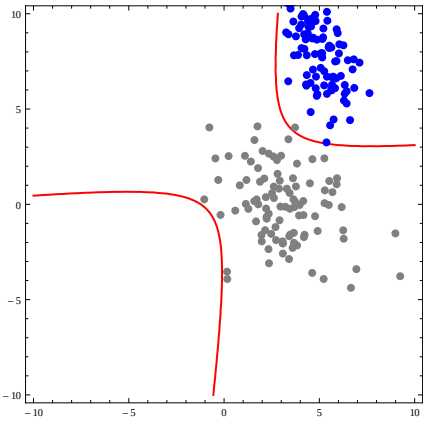
Stohastički gradijentni spust razlikuje se od standardnog po tome što (1) ažuriranje težina radimo unutar petlje koja iterira po svim primjerima, (2) ne koristimo linijsko pretraživanje i (3) prije svakog prolaska kroz sve primjere (tzv. **epoha**) slučajno permutiramo primjere, kako se ažuriranje težina ne bi u svakoj epohi radilo na primjerima u istome redoslijedu, što bi smanjilo stohastičnost gradijentnog spusta. 27

▸ **Logistička regresija – stohastički gradijentni spust**

$\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
ponavljaj do konvergencije
  slučajno permutiraj primjere u $\mathcal{D}$
  za $i = 1, \ldots, N$
    $h \leftarrow \sigma(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}^{(i)}))$
    $\Delta\mathbf{w} \leftarrow -(h - y^{(i)})\boldsymbol{\phi}(\mathbf{x}^{(i)})$
    $\mathbf{w} \leftarrow \mathbf{w} + \eta\Delta\mathbf{w}$

5 Regularizirana regresija

Već znamo da strojno učenje ima problema sa presloženim modelima. Ni algoritam logističke regresije nije imun na taj problem. Evo primjera presloženog modela logističke regresije:



Na slici je prikazana granica između klasa u dvodimenzijskome ulaznom prostoru (crvene krivulje) dobivena algoritmom logističke regresije s funkcijom preslikavanja $\boldsymbol{\phi}(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Ovaj je model očito presložen, jer dobivena hipoteza (skoro) savršeno klasificira sve primjere iz skupa za učenje, međutim granica je više nelinearna nego što bi to trebala biti. Konkretno, donji desni dio ulaznoga prostora proglašen je regijom plavih primjera, premda u tom prostoru nema niti jedan plavi primjer.

Znamo da je jedan od načina da spriječimo prenaučenost, ili barem smanjimo mogućnost prenaučenosti, **regularizacija**. Regularizacijom težine modela potiskujemo prema nuli, što ostvarujemo dodavanjem regularizacijskog izraza u empirijsku pogrešku definiranog tako da kažnjava hipoteze s velikim težinama.

Kod logističke regresije, možemo pričati o "trostrukom učinku" regularizacije:

1. Ako je model nelinearan, regularizacijom sprečavamo prenaučenost uslijed pretjerane nelinearnosti (gornji primjer);
2. Ako imamo puno značajki, regularizacijom efektivno smanjujemo broj značajki jer težine potiskujemo prema nuli, čime sprečavamo prenaučenost uslijed previše značajki (ovo vrijedi za L_1-regularizaciju);
3. Specifično za logističku regresiju: Ako je problem linearno odvojiv, sprječavamo "otvrdnjivanje" sigmoide (povećanje njezina nagiba), koje bi inače opet dovelo do prenaučenosti modela.

28

Posljednja točka zaslužuje dodatno objašnjenje. Zašto porast težina $\mathbf{w}$ uzrokuje otvrdnjavanje sigmoide, zašto bi težine uopće rasle, i zašto je to slučaj samo s linearno odvojivim problemima? Prisjetimo se, najprije, da s porastom magnituda težina, $\|\mathbf{w}\|$, raste i skalarni umnožak $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$. To znači da raste argument sigmoidne funkcije, a već smo bili rekli da je efekt toga to da je sigmoida strmija. Ovo se događa samo ako su primjeri linearno odvojivi, jer samo u tom slučaju otvrdnjavanje sigmoide dovodi do smanjenja pogreške unakrsne entropije. U to se možete uvjeriti ako pogledate graf funkcije logističkog gubitka. Što je umnožak $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y$ veći, to je gubitak $L(y, h(\mathbf{x}))$ manji. Prema tome, optimizacijskom je algoritmu u interesu da povećava magnitudu težina $\mathbf{w}$. Dapače, optimizacijski algoritam će htjeti težine $\mathbf{w}$ povećati do beskonačnosti, jer bi tek tada gubitak bio jednak nuli. Također, prisjetite se s našeg prošlog predavanja da množenje vektora težina $\mathbf{w}$ (koji uključuje težinu w_0) nema efekta na položaj i orijentaciju hiperravnine. To znači da optimizacijski algoritam može proizvoljno povećavati težine $\mathbf{w}$, u nastojanju da smanji empirijsku pogrešku, a da to nema nikakvog utjecaja na položaj i orijentaciju hiperravnine. Jedini efekt toga jest da sigmoida otvrdnjava, tj. postaje sličnija funkciji praga. No, tako nešto ne želimo, jer takav model loše generalizira. Također primijetite da ovaj problem nastupa samo kada su primjeri linearno odvojivi; ako oni to nisu, onda povećanje težina $\mathbf{w}$ uzrokuje smanjenje gubitka na točno klasificiranim primjerima, ali istodobno i povećanje gubitka na netočno klasificiranim primjera, što znači da je minimum empirijske pogreške negdje između toga i da algoritam optimizacije neće beskonačno povećavati težine.

U nastavku ćemo se baviti L_2-regularizacijom, jer je nju lako ugraditi u algoritam gradijentnog spusta koji smo već objasnili. Pogledajmo kako. Izraz za empirijsku pogrešku proširujemo s L_2-regularizacijskim izrazom (označen crveno):

29

$$E_R(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \Big(-y^{(i)} \ln h(\mathbf{x}^{(i)}) - (1 - y^{(i)}) \ln \big(1 - h(\mathbf{x}^{(i)})\big) \Big) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

Gradijent ove funkcije po vektoru težina $\mathbf{w}$ je:

$$\nabla_{\mathbf{w}} E_R(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \big(h(\mathbf{x}^{(i)}) - y^{(i)}\big)\boldsymbol{\phi}(\mathbf{x}^{(i)}) + \lambda\mathbf{w}$$

gdje smo iskoristili jednakost $\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}}\mathbf{w}^{\mathrm{T}}\mathbf{w} = 2\mathbf{w}$ iz matričnog diferencijalnog računa. Ažuriranje težina u standardnom (grupnom) gradijentnom spustu onda je:

30

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\Big(\sum_{i=1}^{N} \big(h(\boldsymbol{\phi}(\mathbf{x}^{(i)})) - y^{(i)}\big)\boldsymbol{\phi}(\mathbf{x}^{(i)}) + \lambda\mathbf{w} \Big)$$

Ako regularizacijski izraz izlučimo iz sume, dobivamo sljedeći alternativni zapis pravila za ažuriranje težina:

$$\mathbf{w} \leftarrow \mathbf{w}(1-\eta\lambda) - \eta \sum_{i=1}^{N} \big(h(\mathbf{x}^{(i)}) - y^{(i)}\big)\mathbf{x}^{(i)}$$

gdje pribrojnik $\mathbf{w}(1-\eta\lambda)$ dovodi do efekta **propadanja težina** (engl. *weight decay*). Naime, primijetite da, ako bi drugi pribrojnik bio konstantan, težine bi se u svakom koraku smanjivale proporcionalno s $(1-\eta\lambda)$. Također primijetite da promjena težina ovisi ne samo o faktoru η, već i o broju primjera N: što je N veći, to je veća promjena težina. Zbog toga stopu učenja η treba korigirati u ovisnosti o broju primjera. Za stopu učenja može se koristiti i vrijednost η/N (što bismo ionako dobili da smo funkciju pogreške definirali kao očekivanje funkcije gubitka).

Posljednja napomena ovdje jest da nikako ne smijemo zaboraviti da težinu w_0 ne regulariziramo! Naime, ako bismo regularizirali težinu w_0, ona bi težina w_0 težila prema nuli. Međutim, budući da w_0 određuje udaljenost hiperravnine od ishodišta (ta udaljenost je $-w_0/\|\mathbf{w}\|$), to znači 31
da bi hiperravnina uvijek morala prolaziti kroz ishodište i da ne bismo mogli dobro razdvojiti dvije klase u slučajevima kada granica ne prolazi baš kroz ishodište (a općenito granica ne mora prolaziti kroz ishodište).

Konačno, definirajmo algoritme L_2-regularizirane logističke regresije. Budući da težinu w_0 moramo tretirati posebno, u pseudokodu ćemo oznaku $\mathbf{w}$ koristiti za težine bez w_0, a oznaku $\tilde{\mathbf{w}}$ za proširen vektor težina koji uključuje w_0. Također, koristimo $\tilde{\mathbf{x}}$ za vektor proširen sa "dummy značajkom" $x_0 = 1$. Algoritam L_2-regularizirane logističke regresije optimiziran standardnim gradijentnim spustom je:

▸ **L2-regularizirana logistička regresija – standardni (grupni) gradijentni spust**

1: $\tilde{\mathbf{w}} \leftarrow (0, 0, \ldots, 0)$
2: **ponavljaj** do konvergencije
3: $\quad(\Delta w_0, \Delta\mathbf{w}) \leftarrow (0, 0, \ldots, 0)$
4: $\quad$**za** $i = 1, \ldots, N$
5: $\quad\quad h \leftarrow \sigma\big(\tilde{\mathbf{w}}^{\mathrm{T}}\boldsymbol{\phi}(\tilde{\mathbf{x}}^{(i)})\big)$
6: $\quad\quad(\Delta w_0, \Delta\mathbf{w}) \leftarrow \Delta\mathbf{w} - (h - y^{(i)})\boldsymbol{\phi}(\mathbf{x})$
7: $\quad\eta \leftarrow$ optimum linijskim pretraživanjem u smjeru $\Delta\tilde{\mathbf{w}}$
8: $\quad w_0 \leftarrow w_0 + \eta\Delta w_0$
9: $\quad\mathbf{w} \leftarrow \mathbf{w}(1-\eta\lambda) + \eta\Delta\mathbf{w}$

Jedina suštinska izmjena u odnosu na algoritam neregularizirane logističke regresije je u retku 9, gdje kod ažuriranja težina imamo efekt propadanja težina.

Algoritam L_2-regularizirane logističke regresije optimiziran stohastičkim gradijentnim spustom je sljedeći:

▸ **L2-regularizirana logistička regresija (stohastički gradijentni spust)**

1: $\tilde{\mathbf{w}} \leftarrow (0, 0, \ldots, 0)$
2: **ponavljaj** do konvergencije:
3: $\quad$slučajno permutiraj primjere u $\mathcal{D}$
4: $\quad$za $i = 1, \ldots, N$
5: $\quad\quad h \leftarrow \sigma\big(\tilde{\mathbf{w}}^{\mathrm{T}}\boldsymbol{\phi}(\tilde{\mathbf{x}}^{(i)})\big)$
6: $\quad\quad(\Delta w_0, \Delta\mathbf{w}) \leftarrow \Delta\mathbf{w} - (h - y^{(i)})\boldsymbol{\phi}(\mathbf{x})$
7: $\quad\quad w_0 \leftarrow w_0 + \eta\Delta w_0$
8: $\quad\quad\mathbf{w} \leftarrow \mathbf{w}(1-\eta\lambda) + \eta\Delta\mathbf{w}$

Kao i kod neregularizirane varijante, razlika između standardnog gradijentnog spusta i stohastičkog gradijentnog spusta jest što se ažuiranje težina provodi unutar petlje koja iterira po svim primjerima, što slučajno permutiramo primjere u svakoj epohi, i što ne radimo linijsko pretraživanje.

Sažetak

- Logistička regresija je **diskriminativan klasifikacijski model** s probabilističkim izlazom
- Aktivacijska funkcija je **sigmoidna (logistička) funkcija**
- Algoritam koristi **pogrešku unakrsne entropije**, čija minimizacija odgovara maksimizaciji vjerojatnosti oznaka na skupu označenih primjera
- Optimizacija se provodi **gradijentnim spustom**, a prenaučenost se može spriječiti **regularizacijom**
- Logistička regresija vrlo je dobar klasifikacijski algoritam koji nema nedostatke koje imaju klasifikacija linearnom regresijom i perceptron

Bilješke

[1] Ovo je terminološka mina: **logistička regresija**, naime, nije algoritam regresije, nego klasifikacijski algoritam. Vjerojatno se pitate tko je tu lud i zašto je to tako. Odgovor je da ova terminološka nekonzistentnost postoji samo iz perspektive strojnog učenja, gdje se logistička regresija doista dominantno koristi kao algoritam za binarnu klasifikaciju, s izlazima 0 i 1. Međutim, logistička je regresija prvenstveno statistički model: u statistici je logistička regresija tek jedan u nizu modela za analizu zavisnosti između nezavisnih varijabli i zavisne varijable. U statistici nema terminološke nekonzistentnost jer tamo logistička regresija doista *jest* regresijski model s izlazima u intervalu $(0, 1)$, gdje se onda tek dodatnim uvođenjem praga (tipično postavljenim na 0.5) dobiva klasifikacijski algoritam. Ovaj odgovor daje više detalja: `https://stats.stackexchange.com/a/127044/93766`. Standardne reference iz primijenjene statistike koje se bave logističkom regresijom su [Harrell Jr, 2015] i [Hosmer Jr et al., 2013].

[2] Ovdje imamo još jednu terminološku minu: **poopćeni linearni modeli** (engl. *generalized linear models*) nisu isto što i **opći linearni model** (engl. *general linear model*). Poopćeni linearni modeli poopćenje su linearne regresije kod kojega je skalarni umnožak $\mathbf{w}^{\mathrm{T}}\mathbf{x}$ omotan nekom aktivacijskom funkcijom f. U statističkom smislu, to su modeli kod kojega je pogreška zavisne varijable poopćena sa normalne distribucije na neku drugu distribuciju (npr. Poissonovu, Bernoullijevu), gdje $f(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ određuje parametar μ srednje vrijednosti te distibucije. Logistička regresija, kojom se bavimo danas, je poopćeni linearni model gdje je pogreška zavisne varijable modelirana Bernoullijevom distribucijom.
S druge strane, opći linearan model je sažet način zapisa više različitih statističkih modela temeljenih na višestrukoj linearnoj regresiji (linearna regresija, ANOVA, ANCOVA, MANOVA, MANCOVA, t-test i F-test). Mi se nećemo baviti općim linearnim modelom.
U literaturi se za poopćene linearne modele ponekad umjesto kratice GLM koristi kratica GZLM ili GLiM, kako bi ih se razlikovalo od općih linearnih modela za koje se isto koristi kratica GLM. Poopćene linearne modele kao unifikaciju različitih statističkih modela predložili su 1972. godine statističari John Nedler i Rober Wedderburn [Nelder and Wedderburn, 1972]. U intervjuu 2003. godine za časopis *Statistical Science*, Nedler je priznao da ime "poopćeni linearni modeli" možda i nije bila najsretnija opcija zbog konfuzije s općim linearnim modelima, no sada je gotovo. Standardne reference za poopćene linearne modele su [McCullagh and Nelder, 1989] i [Dobson and Barnett, 2018].

[3] Priznajem da sam ovaj prijevod izmislio.

[4] **Logistička funkcija** ili **logistička krivulja** ili **S-krivulja** predložena je prvi puta još 1838. godine kao model rasta populacije u radovima belgijskog matematičara Pierra Françoisa Verhulsta. Naziv

"logistički" osmislio je upravo Verhulst, kao kontrast od "logaritamski". Naziv, inače, nema nikakve veze sa "logistikom" (u smislu znanja i sredstava za strateško vojno ili slično organizirano djelovanje). Logistička je krivulja zapravo vrlo značajna jer dobro opisuje mnoge prijelazne pojave o prirodi, od aktivacije neurona i magnetizacije željeza do topljenja leda i promjena znanstvenih paradigmi. Lijep opis toga možete naći u knjizi Pedra Domingosa [Domingos, 2015], koju smo bili spomenuli u uvodnome predavanju, i to u čevrtome poglavlju naslovljenome "Najvažnija krivulja na svijetu". Domingos povlači zgodnu paralelu između logističke krivulje i poznatog citata iz Hemingwayevog romana *I sunce se rađa* (*The Sun Also Rises*), u kojemu vječito pijan Mike Cambell odgovara na pitanje o tome kako je bankrotirao: "Dva načina. Postupno, a onda iznenada." (*"Two ways. Gradually and then suddenly"*). Doista, mnoge fazne tranzicije u našim životima odvijaju se upravo takvom dinamikom kakvu opisuje logistička krivulja.

5 Naravno, činjenica da je izlaz modela u intervalu $[0, 1]$ (ili intervalu $(0, 1)$) ne znači automatski da ga možemo tumačiti kao vjerojatnost. Možemo li takav izlaz legitimno interpretirati kao vjerojatnost ovisi o tome kako definiramo vjerojatnost i koje distribucijke pretpostavke su ugrađene u model. O tome više u drugom dijelu predmeta.

6 Već smo rekli da je vjerojatnost $P(\mathbf{y}|\mathbf{X})$ zapravo **izglednost** (engl. *likelihood*) parametara $\mathbf{w}$, koji preko modela $h(\mathbf{x}^{(i)}; \mathbf{w})$ definiraju srednju vrijednost distribucije oznaka $y^{(i)}$. No, nećemo ovdje uvoditi taj pojam, već to ostavljamo za temu broj 14.

7 U izrazu $P(y|\mu)$, y je slučajna varijabla, a μ je parametar distribucije i on nije slučajna varijabla. Ovo može zbunjivati jer, npr., u izrazu $P(y|x)$ su i y i x slučajne varijable. Moguće su i mješovite situacije, npr. u izrazu $P(y|\mathbf{x}, \mathbf{w})$ izrazi y i $\mathbf{x}$ su slučajne varijable, a $\mathbf{w}$ je parametar. U tom smislu, bilo bi bolje pisati $P(y; \mu)$ odnosno $P(y|\mathbf{x}; \mathbf{w})$, kako bi se parametri jasno razdvojili od slučajnih varijabli. Nažalost, to nije praksa u literaturi iz strojnog učenja i najbolje je razviti imunost na ovu notacijsku nedosljednost. Dobra je disciplina dati si zadatak za svaki izraz pojasniti tipove svih izraza koji se u njemu javljaju, tako da za svaki izraz znate reći radi li se o slučajnoj varijabli, konstanti ili parametru.

8 Naziv reflektira činjenicu da pogreška doista odgovara **unakrsnoj entropiji** iz teorije informacije, pri čemu je prava distribucija distribucija oznaka y, a procijenjena distribucija ona koju daje model $h(\mathbf{x})$. Naime, unakrsna entropija (za diskretne vjerojatnosne distribucije) definirana je kao:

$$H(p, q) = -\sum p(x) \ln q(x)$$

te ona odgovara prosječnome broju bitova potrebih za kodiranje događaja čija je vjerojatnost nastupanja određena distribucijom $p(x)$, a shema kodiranja optimirana je za distribuciju $q(x)$.

9 Naime, vrijedi:

$$h(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))} \qquad 1 - h(\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}$$

pa onda:

$$\begin{aligned} L(y, h(\mathbf{x})) &= -y \ln h(\mathbf{x}) - (1 - y) \ln(1 - h(\mathbf{x})) \\ &= y \ln\big(1 + \exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))\big) + (1 - y) \ln\big(1 + \exp(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))\big) \end{aligned}$$

Sada želimo da lijevi izraz bude jednak nuli za $y = -1$, a desni izraz da bude jednak nuli za $y = +1$. Zatim uočavamo da se lijevi i desni izraz razlikuju jedino po predznaku. To nam onda omogućava da ih stopimo u jedan izraz, ovako:

$$\ln\big(1 + \exp(-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y)\big)$$

i to je upravo ono što smo željeli: gubitak iskazan kao funkcija od $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})y$.

10 Dakle, formalno gledano, **gubitak unakrsne entropije** (engl. *cross-entropy loss*) i **logistički gubitak** (engl. *logistic loss*) nije jedno te isto. Gubitak unakrsne entropije je funkcija dviju varijabli, $L(y, h(\mathbf{x}))$, definirana za $y = \{0, 1\}$, dok je logistički gubitak funkcija jedne varijable, $L(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x})y)$, definirana za $y = \{-1, +1\}$. Međutim, obje ove funkcije jesu funkcije gubitka algoritma logističke

regresije i razlika je samo pitanje konvencije. Uglavnom je iz konteksta uvijek jasno na koju se varijantu misli, i čini se da se ljudi u literaturi oko toga ne uzbuđuju previše.

11 Kod nekih algoritama ima čak smisla krenuti najprije od optimizacijskog postupka, a tek onda iz njega izvesti funkciju gubitka i funkciju pogreške. Tako ćemo raditi kod stroja potpornih vektora (SVM), gdje ćemo najprije definirati otpimizacijski problem maksimalne margine kao problem kvadratnog programiranja, a tek onda iz toga izvesti funkciju pogreške i funkciju gubitka SVM-a.

12 Gradijent funkcije pogreške unakrsne entropije je

$$\nabla E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \big(h(\mathbf{x}^{(i)}) - y^{(i)}\big)\boldsymbol{\phi}(\mathbf{x}^{(i)})$$

U ovom izrazu težine $\mathbf{w}$ su u nelinearnom odnosu sa E, i to dovodi do toga da rješenje jednadžbe

$$\nabla E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^{N} \big(h(\mathbf{x}^{(i)}) - y^{(i)}\big)\boldsymbol{\phi}(\mathbf{x}^{(i)}) = 0$$

ne postoji zatvorenoj formi. Prisjetimo se da kod linearne regresije nemamo taj problem: tamo rješenje postoji u zatvorenoj formi, no to je upravo zato jer ne postoji nelinearnost (nema aktivacijske funkcije). Čini se da je postojanje rješenja optimizacijskoj problema poopćenog linearnog modela u zatvorenoj formi više iznimka nego pravilo, v. `https://stats.stackexchange.com/a/37640/93766`.

13 Kod logističke regresije, glavna alternativa gradijentnom spustu i iz njega izvedenih varijanti jest optimizacija drugog reda, konkretno **Newtonov postupak** i njegova aproksimativna varijanta, **kvazi-Newtonov postupak**, a posebice računalno učinkovita implementacije kvazi-Newtonovog postupka pod nazivom **LM-BFSG** (engl. *limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm*). O svemu tome pričat ćemo (doduše bez previše detalja) idući put.

14 Standardna referenca za **konveksnu optimizaciju** je [Boyd et al., 2004]. Općenito, konveksna se optimizacija bavi problemima koji se mogu formalizirati kao:

$$\begin{aligned} \text{minimiziraj} \quad & f_0(\mathbf{x}) \\ \text{tako da} \quad & f_i(\mathbf{x}) \leqslant b_i, \quad i = 1, \ldots, m \end{aligned}$$

gdje su $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ konveksne funkcije. Funkcija $f(\mathbf{x})$ je **funkcija cilja** (ili **kriterijska funkcija**), a $f_i(\mathbf{x}) \leqslant b_i$ su ograničenja jednakosti ili ograničenja nejednakosti. Ako imamo ograničenja, onda govorimo o **optimizaciji s ograničenjima** (engl. *constrained optimization*). Više o tome u predavanju o SVM-u.

15 Po nekima, područje strojnog učenja previše je, gotovo religiozno, privrženo konveksnoj optimizaciji, premda su mnogi stvarni problemi u strojnom učenju nekonveksni optimizacijski problemi. To je sasvim sigurno slučaj u **dubokom učenju**, kod kojega više slojeva nelinearnih aktivacijskih funkcija dovode do toga da je funkcija pogreško visoko nelinearna i nekonveksna. Za pregled nekonveksnih pristupa optimizaciji u strojnom učenju, pogledajte [Jain and Kar, 2017].

16 **Konveksnost funkcije** se, osim izravno na temelju definicije, što je u nekim slučajevima teško, može dokazati na nekoliko načina. Npr., dvostruko diferencijabilna funkcija je konveksna u zadanom intervalu ako i samo ako je njezina druga derivacija u tom intervalu nenegativna. U praksi je, međutim, konveksnost lakše dokazati tako da se funkcija napiše pomoću funkcija za koje je već dokazano da su konveksne, korištenjem operacija za koje je dokazano da čuvaju konveksnost (engl. *convexity preserving operations*). Vidjeti, npr., poglavlje 3 u [Boyd et al., 2004]. U poglavlju 9 iste knjige opisuje se gradijentni spust u kontekstu optimizacije konveksnih funkcija.

17 Nemojte brkati **globalnu konveregenciju** i **globalni minnimum**. Globalna konvergencija znači samo da se pretraga zaustavlja nakon konačnog broja koraka, ali ne mora se nužno zaustaviti u točki globalnog minimuma.

18 Malo više o **linijskom pretraživanju** (ili **pretraživanju po pravcu**) možete pročitati na `https://en.wikipedia.org/wiki/Line_search`. Linijsko pretraživanje može biti egzaktno (ekplicitna minimizacije funkcije $F(\mathbf{x}+\eta\Delta\mathbf{x})$) ili približno. Algoritam za približno pretraživanje je **linijsko pretraživanje**

s povratkom (engl. *backtracking line search*). Riječ je o iterativnom algortimu koji u svakoj iteraciji smanjuje korak na temelju lokalnog gradijenta funkcije koja se minimizira, na temelju tzv. Armijo-Goldsteinovog uvjeta (pravila). Više na `https://en.wikipedia.org/wiki/Backtracking_line_search`. Detaljni opis linijskog pretraživanja možete naći u poglavlju 3 knjige [Nocedal and Wright, 2006].

19 Preuzeto sa `http://fourier.eng.hmc.edu/e176/lectures/NumericalMethods/node18.html`.

20 Pregled optimizacijskih postupaka temeljenih na gradijentnom spustu s prilagodivom stopom učenja možete naći u [Ruder, 2016] i [Goodfellow et al., 2016] (poglavlje 8).

21 Ovakvi se problemi u teoriji optimizacije nazivaju **problemi konačne sume** (engl. *finite sum problems*). V. `http://www.stat.cmu.edu/~ryantibs/convexopt-F18/scribes/Lecture_9.pdf`

22 Više o **stohastičkom gradijentnom spustu** možete pročitati u poglavlju 7 iz [Deisenroth et al., 2020]. Matematički rigorozniji tretman možete naći u poglavlju 13 u [Sra et al., 2012] i u [Bottou and Bousquet, 2008].

23 Situacija u kojoj ćemo sigurno koristiti stohastički a ne standardni gradijentni spust jest ona kada nemamo na raspolaganju odmah sve primjere, nego učimo postepeno, kako dolaze novi primjeri. Takav način učenja nazivamo **online učenje**. Kod online učenja postoji **tok podataka** i trebamo dinamički trenirati klasifikator kako dolaze novi podatci. Pregled metoda za online učenje možete naći u [Gama, 2012] i [Benczúr et al., 2018].

24 U dubokom učenju površina funkcije pogreške redovito je vrlo složena i ima mnogo lokalnih minimuma. Također, redovito raspolažemo povećom količinom označenih primjera. To dvoje u kombinaciji govori u prilog korištenja gradijentnog spusta s minigrupama umjesto standardnog gradijentnog spusta. Optimizacijski postupci u dubokom strojnom učenju uglavnom su varijante gradijentnog spusta s adaptivnom (prilagodivom) stopom učenja. Izvrstan pregled raznih varijanti algoritma gradijentnog spusta, od kojih se mnoge standardno koriste u dubokom učenju, možete naći u [Ruder, 2016].

25 Moderne implementacija algoritma logističke regresije tipično koriste algoritme brzog **stohastičkog inkrementalnog gradijentnog spusta**, npr. SAG (engl. *stochastic average gradient*) [Schmidt et al., 2017], kod koje se konvergencija ubrzava pamćenjem gradijenta iz prethodne iteracije, i naprednija varijanta tog algortima pod nazivom SAGA [Defazio et al., 2014]. Pregled metoda inkrementlanog gradijentnog spusta možete naći u `http://niaohe.ise.illinois.edu/ie598/IE598_BigDataOpt_lecturenotes_fall2016.pdf` (lekcija 23).

26 Vidi `https://stats.stackexchange.com/q/321592/93766`.

27 Empirijski je utvrđeno da stohastički gradijentni spust sa slučajnom permutacijom redoslijeda primjera u svakoj epohi radi bolje nego stohastički gradijentni spust s fiksnim poretkom primjera [Bottou, 2009]. Međutim, razlozi za to dugo nisu bili jasni. Tek su nedavno napravljene teorijske analize konvergencije i ograda pogreške stohastičkog gradijentnog spusta koje daju teorijsko opravdanje za učinkovitost slučajne permutacije kod stohastičkog gradijentnog spusta [Ying et al., 2017, Gürbüzbalaban et al., 2019, Safran and Shamir, 2020].

28 Ovaj fenomen (da linearno odvojivi problemi mogu dovesti do parametara čije vrijednosti teže k beskonačnosti, tj. da minimizator pogreške ne postoji) u statistici je poznat pod nazivom **Hauck-Donnerov efekt**. Taj je efekt relevantan kada se želi napraviti statističko zaključivanje o intervalima pouzdanosti parametara ili veličini učinka. Više ovdje: `https://stats.stackexchange.com/q/254124/93766`.

29 O algoritmu optimizacije za L_1-regulariziranu logistička regresiju možete pročitati u [Lee et al., 2006] i [Koh et al., 2007]. Već spomenuti optimizacijski algoritam SAGA [Defazio et al., 2014], koji se danas koristi za logističku regresiju, podržava L_2- i L_1-regularizaciju.

30 Ne baš. Jednakost iz matričnog diferencijalnog računa na koju se ovdje pozivamo glasi:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{x}^{\mathrm{T}}(\mathbf{A} + \mathbf{A}^{\mathrm{T}})$$

pa uz $\mathbf{A} = \mathbf{I}$ imamo $\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}}\mathbf{w}^{\mathrm{T}}\mathbf{w} = 2\mathbf{w}^{\mathrm{T}}$. Stoga bi gradijent regularizacijskog izraza trebao biti $\lambda\mathbf{w}^{\mathrm{T}}$ a ne $\lambda\mathbf{w}$. Međutim, pišemo $\lambda\mathbf{w}$, jer je i prvi pribrojnik zapravo već transponiran (naime, izravna primjena matričnog diferencijalnog računa na gradijent funkcije logističkog gubitka zapravo daje $(h(\mathbf{x}) - y)\phi(\mathbf{x})^{\mathrm{T}}$). Na koncu, mi uvijek baratamo vektor stupcima, pa i gradijent treba biti vektor stupac, makar to značilo naknadno transponiranje nakon deriviranja.

31 Da je udaljenost hiperravnine od ishodiška jednaka $-w_0/\|\mathbf{w}\|$ pokazali smo u bilješci 3 u predavanju 5.

Literatura

A. A. Benczúr, L. Kocsis, and R. Pálovics. Online machine learning in big data streams. *arXiv preprint arXiv:1802.05872*, 2018.

L. Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science, Paris*, 2009.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

A. J. Dobson and A. G. Barnett. *An introduction to generalized linear models*. CRC press, 2018.

P. Domingos. *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.

J. Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming*, pages 1–36, 2019.

F. E. Harrell Jr. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015.

D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

P. Jain and P. Kar. Non-convex optimization for machine learning. *arXiv preprint arXiv:1712.07897*, 2017.

K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research*, 8(Jul):1519–1555, 2007.

S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient l˜ 1 regularized logistic regression. In *Aaai*, volume 6, pages 401–408, 2006.

P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman and Hall, 1989.

J. A. Nelder and R. W. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.

J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

I. Safran and O. Shamir. How good is sgd with random shuffling? In *Conference on Learning Theory*, pages 3250–3284. PMLR, 2020.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. MIT Press, 2012.

B. Ying, K. Yuan, S. Vlaski, and A. H. Sayed. On the performance of random reshuffling in stochastic learning. In *2017 Information Theory and Applications Workshop (ITA)*, pages 1–5. IEEE, 2017.