7. Logistička regresija II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.1

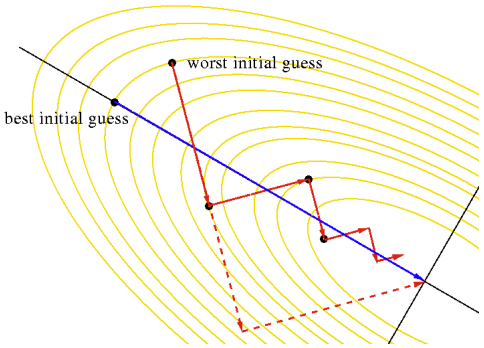
Prošli put upoznali smo se s algoritmom **logističke regresije**. Definirali smo model i zatim izveli funkciju pogreške unakrsne entropije kao negativan logaritam vjerojatnosti oznaka u skupu za učenje. Ustanovili smo da minimizacija te pogreške nema rješenje u zatvorenoj formi, pa smo se okrenuli iterativnim postupcima. Razmotrili smo najjednostavniji takav postupak, algoritam **gradijentnog spusta**, te smo ga primijenili na logističku regresiju, i to u standardnoj (grupnoj) i stohastičkoj izvedbi. Na koncu smo pričali o **regularizaciji**, i to konkretno L_2-regularizaciji, koju smo vrlo jednostavno ugradili u optimizacijski postupak.

Danas ćemo još malo pričati o logističkoj regresiji. Prvo, razmotrit ćemo neke alternative gradijentnom spustu, koje su učinkovitije (čitaj: brže) od gradijentnog spusta. Drugo, razmotrit ćemo proširenje binarne logističke regresije na **višeklasnu logističku regresiju**. Treće, malo ćemo se osvrnuti na dosada naučeno i pogledati što svi ti modeli imaju zajedničko i kako ih možemo poopćiti. Konačno, pričat ćemo o **adaptivnim baznim funkcijama** kao načinu da iz podataka naučimo funkciju preslikavanja iz ulaznog prostora u prostor značajki, umjesto da tu funkciju definiramo ručno.

Za razliku od prethodnih predavanja, ovo će se predavanje uglavnom zadržati na razini upoznavanja, jer za više od toga nemamo vremena. Cilj mi je dati vam dovoljno informacija da znate samostalno gdje kopati dalje, ako će vam to trebati.

1 Alternative gradijentnom spustu

Prošli smo put ustanovili da gradijentni spust treba kombinirati s linijskim pretraživanjem, jer u suprotnom nemamo zajamčenu **globalnu konvergenciju**. To znači da se, ovisno o početnoj točki pretraživanja, može dogoditi da gradijentni spust ne konvergira već da divergira (efektivno se umjesto spuštanja počinje podizati). Linijsko pretraživanje sprječava da se to dogodi. Međutim, linijsko pretraživanje može rezultirati krivudavim ("cik-cak") spustom. Prisjetimo se slike koju smo bili komentirali prošli put:



Plava trajektorija odgovara najboljem scenariju, a crvena najgorem scenariju za gradijentni spust s linijskim pretraživanjem. Scenarij ovisi o odabiru početne točke pretraživanja. Vidimo da se može dogoditi da spust bude brlo krivudav, što znači da će optimizacija konzumirati puno iteracija. Očito, problem nastaje zbog toga što smjer spuštanja nije toliko dobar koliko bi

mogao biti. Zamislite da se spuštate u neku jamu iz početne točke za crvenu trajektoriju. Teško da biste se spuštali baš po označenom pravcu. Sila teža bi vas vukla da se spuštate strmijim smjerom (tj. pod manjim kutom u odnosu na plavu liniju). Da je to tako zaključujemo na temelju zakrivljenosti izokontura (odnosno površine niz koju se spuštamo). Drugim riječima, zakrivljenosti površine daje nam, pored gradijenta, dodatnu informaciju o tome gdje se bi se mogao nalaziti minimum (barem kada je riječ o konveksnim funkcijama).

Gornja opažanja vrijede za standardni (grupni) gradijent. Kod stohastičkog gradijentnog spusta tipično ne koristimo linijsko pretraživanje. No, tamo će spust ionako dosta krivudati, budući da se svaki korak spusta radi na temelju gradijenta izračunatog za jedan pojedinačni primjer. U nastavku se fokusiramo na nestohastički, dakle grupni gradijentni spust.

Na temelju gornjeg razmatranja možemo zaključiti da bismo grupni gradijentni spust mogli unaprijediti, ako bismo u obzir uzeli ne samo nagib (gradijent) nego i zakrivljenost (promjenu gradijenta, tj. drugu derivaciju) funkcije pogreške. Takvi optimizacijski postupci zovu se **optimizacija drugog reda**, za razliku od optimizacijskih postupaka prvog reda, kao što je to gradijentni spust. Osnovni postupak drugog reda jest **Newtonov postupak**. 1

1.1 Newtonov postupak

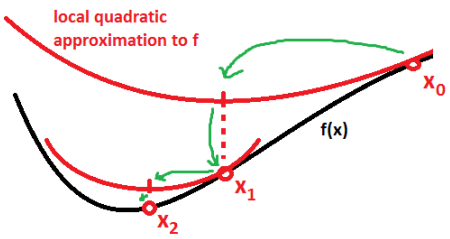
Razmotrimo minimizaciju funkcije $f(\mathbf{x})$. Ažuriranje parametara kod gradijentnog spusta bilo je:

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f(\mathbf{x})$$

Ako uvedemo indeks za interacije, onda to možemo napisati kao jednadžbu:

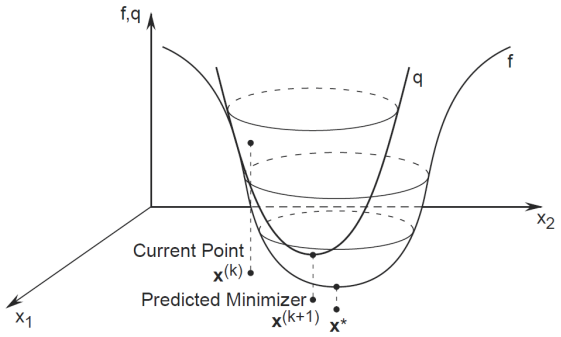
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

Ideja kod Newtonovog algoritma jest da u točki $\mathbf{x}_t$ (trenutačni minimum) napravimo kvadratnu aproksimaciju funkcije $f(\mathbf{x})$, i da onda skočimo u minimizator te kvadratne aproksimacije (taj minimizator je analitički poznat). Ako je f funkcija jedne varijable, to izgleda ovako:



Crna krivulja je funkcija $f(x)$ koju minimiziramo. Krećemo iz točke x_0. U toj točki radimo kvadratnu aproksimaciju funkcije f, čime dobivamo parabolu koja je tangencijalna s funkcijom f u točki x_0. Zatim se pretraga pomiče u točku koja minimizira kvadratnu aproksimaciju od f. Na slici je to točka x_1. U toj točki opet radimo kvadratnu aproksimaciju funkcije f te se pomičemo u točku koja minimizira tu aproksimaciju. Postupak se tako iterativno ponavlja dok ažuriranje nije dovoljno malo.

Postupak funkcionira identično za funkciju više varijabli, tj. u višedimenzijskome prostoru parametara. U dvodimenzijskome prostoru to izgleda ovako:



Dakle, ideja je da korak napravimo točno tako da sletimo u minimum kvadratne aproksimacije funkcije. Ovo će raditi dobro ako je $f(\mathbf{x})$ **konveksna funkcija**, a to kod nas (kod logističke regresije) jest slučaj.

Pozabavimo se sada tehnikalijama. Trebamo napraviti kvadratnu aproksimaciju funkcije u nekoj točki. Kako to napraviti? Prisjetimo se, iz matematičke analize, da svaku diferencijabilnu funkciju $f(x)$ možemo u nekoj zadanoj točki a napisati u obliku reda potencija. Preciznije, svaku diferencijabilnu funkciju možemo razviti u **Taylorov red** u okolini točke a, ovako:

$$\begin{aligned} f(x) &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots \\ &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \end{aligned}$$

Budući da nama treba samo kvadratna aproksimacija, uzeti ćemo samo prva tri člana Taylorovog reda. To je onda **Taylorova aproksimacija drugog reda**: 2

$$f(x) \approx f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2$$

Primijetite da je ovo sada aproksimacija; jednakost između $f(x)$ na lijevoj strani i reda na desnoj strani vrijedi samo kada je red beskonačan.

Naša funkcije pogreške, $E(\mathbf{w}|\mathcal{D})$, je funkcije više varijabli (vektora $\mathbf{w}$). Za funkciju više varijabli $f(\mathbf{x})$, kvadratni razvoj oko točke $\mathbf{x}_t$ je:

$$f(\mathbf{x}) \approx f_{\text{quad}}(\mathbf{x}) = f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^{\mathrm{T}}(\mathbf{x}-\mathbf{x}_t) + \frac{1}{2}(\mathbf{x}-\mathbf{x}_t)^{\mathrm{T}}\mathbf{H}_f(\mathbf{x}_t)(\mathbf{x}-\mathbf{x}_t)$$

gdje je $\mathbf{H}_f(\mathbf{x}_t)$ **Hesseova matrica** (engl. *Hessian matrix*) funkcije $f(\mathbf{x})$ u točki $\mathbf{x}_t$. Hesseova matrica je kvadratna matrica dimenzija $n \times n$ parcijalnih derivacija drugog reda funkcije $f : \mathbb{R}^n \to \mathbb{R}$, odosno funkcije koja n-dimenzijske vektore preslikava u skalare. Hesseova matrica definirana je ovako: 3

$$\mathbf{H}_f = \nabla\nabla f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

to jest, element Hesseove matrice definiran je s: 4

$$(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Primijetite da je Hesseova matrica **simetrična matrica**, budući da redoslijed parcijalnih derivacija ne utječe na rezultat (komutativnost).

Iz ovog raspisa kvadratne funkcije f_{quad} sada se može izvesti **minimum** te funkcije, i to je upravo korak koji želimo napraviti pri spustu (izvod preskačemo). Ažuriranje parametara onda je: [5]

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \mathbf{H}_f(\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t)$$

Ako je Hesseova matrica egzaktno izračunata (tj. nije aproksimacija), onda možemo staviti $\eta = 1$, jer ćemo tako napraviti korak točno u minimum kvadratne aproksimacije.

Vidimo da će za Newtonov optimizacijski postupak biti potrebno računati **inverz Hesseove matrice** funkcije $f(\mathbf{x}_t)$ u točki $\mathbf{x}_t$. Općenito, matrica $\mathbf{H}_f(\mathbf{x}_t)$ je **pozitivno semidefinitna** ($\mathbf{x}^{\mathrm{T}}\mathbf{H}_f(\mathbf{x}_t)\mathbf{x} \geqslant 0$ za svaki ne-nul vektor $\mathbf{x}$) ako i samo ako je funkcija $f(\mathbf{x})$ **konveksna**. Međutim, da bi imala inverz, matrica $\mathbf{H}_f(\mathbf{x}_t)$ trebala bi biti **pozitivno definitna** ($\mathbf{x}^{\mathrm{T}}\mathbf{H}_f(\mathbf{x}_t)\mathbf{x} > 0$ za svaki ne-nul vektor $\mathbf{x}$). Ako je matrica $\mathbf{H}_f(\mathbf{x}_t)$ pozitivno semidefinitna, ali nije pozitivno definitna, onda nema inverz i tada ne možemo primijeniti Newtonov postupak. [6]

1.2 Newtonov postupak za logističku regresiju

Primijenimo sada Newtonov postupak na logističku regresiju. Pravilo za ažuriranje težina je:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}_E^{-1}(\mathbf{w}) \nabla_{\mathbf{w}} E(\mathbf{w}|\mathcal{D}) \qquad (\text{uz } \eta = 1)$$

Lako se može izvesti da je Hesseova matrica za pogrešku unakrsne entropije E sljedeća: [7]

$$\mathbf{H}_E = \mathbf{\Phi}^{\mathrm{T}} \mathbf{S} \mathbf{\Phi}$$

gdje je $\mathbf{S} = \operatorname{diag}\big(h(\mathbf{x}^{(i)})(1 - h(\mathbf{x}^{(i)}))\big)$, tj. dijagonalna matrica koja na dijagonali ima derivacije prvog reda logističkog izlaza $h(\mathbf{x})$ za svaki od N primjera iz skupa za učenje. Za svaki slučaj, provjerimo kompatibilnost matrica koje ovdje množimo: $((m+1) \times N) \cdot (N \times N) \cdot (N \times (m+1))$. Sve je u redu: dimenzija matrice $\mathbf{H}_E$ je $(m+1) \times (m+1)$, gdje je m broj značajki u prostoru značajki (dakle, nakon preslikavanja).

Vratimo se nakratko na problem izračunavanja inverza Hesseove matrice $\mathbf{H}_E(\mathbf{w})$. Kako smo već rekli, $\mathbf{H}_E(\mathbf{w})$ je pozitivno semidefinitna ako i samo ako je funkcija E konveksna. Pogrešna unakrsne entropije je konveksna, pa će dakle matrica $\mathbf{H}_E(\mathbf{w})$ za logističku regresiju uvijek (u svakoj točki $\mathbf{w}$) biti pozitivno semidefinitna. Ali to i dalje ne znači da uvijek ima inverz. Srećom, može se pokazati da, budući da vrijedi rastav $\mathbf{H}_E = \mathbf{\Phi}^{\mathrm{T}} \mathbf{S} \mathbf{\Phi}$, to $\mathbf{H}_E(\mathbf{w})$ mora biti pozitivno definitna. Drugim riječima, Hesseova matrica unakrsne entropije imati će inverz u svakoj točki $\mathbf{w}$, pa ćemo dakle za optimizaciju parametara logističke regresije uvijek moći primijeniti Newtonovu metodu. Ipak, zbog multikolinearnosti, $\mathbf{H}_E(\mathbf{w})$ može biti loše kondicionirana i rješenje će tada biti nestabilno, što znači da trebamo ili odabrati linearno nezavisan podskup značajki ili provesti regularizaciju (vidjet ćemo uskoro kako). [8]

Ako bismo sada uvrstili $\mathbf{\Phi}^{\mathrm{T}} \mathbf{S} \mathbf{\Phi}$ umjesto $\mathbf{H}_E$ u gornje pravilo za ažuriranje težina i malo presložili izraz, dobili bismo **algoritam najmanjih kvadrata s iterativnim ažuriranjem težina** (engl. *iteratively reweighted least squares, IRLS*). Nećemo u detalje; dovoljno je da znate da se taj algoritam koristi za bržu optimizaciju za logističku regresiju, i da je to zapravo primjena Newtonovog postupka, koji je optimizacija drugog reda. [9]

1.3 Kvazi-Newtonov postupak

Problem s Newtonovim postupkom (a tako i s IRLS-om) jest što izračun Hesseove matrice (a pogotovo njezina inverza) može biti skup (pogotovo ako je m, broj dimenzija prostora značajki, velik). Primijetite da u svakom koraku optimizacije moramo izračunati Hesseovu matricu i njezin inverz. Lako može biti da se više isplati raditi gradijentni spust, pa makar taj krivudao!

Alternativa jest da se, umjesto egzaktne Hesseove matrice, izračunava njezina aproksimacija. To rade tzv. **kvazi-Newtonovi postupci**. Ti postupci u svakom koraku spusta aproksimiraju Hesseovu matricu (ili njezin inverz) pomoću vektora gradijenta (iz trenutačnog i prethodnog

koraka). Najpoznatija takva metoda jest algoritam **BFSG**. Opet, nećemo u detalje, trebate samo znati da to postoji. [10]

Drugi problem je što, čak i ako aproksimiramo Hesseovu matricu, njezino pohranjivanje u memoriju može biti problematično, jer je ona dimenzija $(m+1)\times(m+1)$. U tom slučaju može se napraviti "komprimiranje" matrice metodom **aproksimacije matricom nižeg ranga**. Algoritam koji tako funkcionira jest **ograničen BFSG** (engl. *limited BFSG,* **L-BFSG**). Opet, dovoljno je samo da znate da takav algoritam postoji. [11] [12]

1.4 Newtonov postupak s regularizacijom

Znamo već koje su prednosti regularizacije, a prošli smo puta pričali i o dodatnom značaju regularizacije kod logističke regresije (kod optimizacije parametara za linearno odvojive probleme imamo $\|\mathbf{w}\| \to \infty$ i logistička se regresija lako prenauči). Proširenje Newtonovog postupka na regulariziranu pogrešku unakrsne entropije je jednostavno. Pogledajmo L_2-**regularizaciju**. Dodavanje L_2-regularizacijskog izraza na gradijent smo već imali:

$$\nabla E_R(\mathbf{w}|\mathcal{D}) = \nabla E(\mathbf{w}|\mathcal{D}) + \lambda\mathbf{w}$$

Slično imamo i za Hesseovu matricu:

$$\mathbf{H}_R = \mathbf{H} + \lambda\mathbf{I}$$

gdje je $\lambda\mathbf{I}$ dijagonalna matrica s regularizacijskim faktorom λ na dijagonali (osim na gornjoj lijevoj poziciji, jer težinu w_0 ne regulariziramo).

Vidimo, dakle, da se L_2-regularizacija lako ugradi u postupak gradijentnog spusta ili u Newtonov postupak, kao što se uostalom i lako ugradila u optimizacijski postupak najmanjih kvadrata kod linearne regresije. Sve u svemu, L_2-regularizacija je jedna pitoma zvjerka.

Što je s L_1-regularizacijom? S njome se do sada uopće nismo bavili, a nećemo ni sada. Čisto informativno, trebamo znati da se tu stvari očekivano kompliciraju zbog toga što L_1-norma nije diferencijabilna, pa ne možemo računati gradijent. Umjesto toga računamo **podgradijent** (engl. *subgradient*). Zatim tipično koristimo **koordinatni spust** (engl. *coordinate descent*), gdje optimiramo varijablu po varijablu (dimenziju po dimenziju), ili koristimo **proksimalne** ili **projekcijske** optimizacijske postupke. Za sada samo trebate znati da je optimizacija L_1-regularizirane logističke regresije moguća, da za to postoje algoritmi i da su implementirani u standardnim alatima. [13]

2 Višeklasna logistička regresija

Prošli put bavili smo se **binarnom logističkom regresijom**: klasificirali smo u klase $y = 1$ i $y = 0$. Kako bismo dobili vjerojatnosti, za aktivacijsku funkciju koristili smo sigmoidu:

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

No, što ako imamo više od dvije klase, tj. $K > 2$? Mogli bismo primijeniti dekompozicijsku shemu OVO ili OVR, ali problem je što se vjerojatnosti za pojedinačne klase ne bi zbrajale u 1. Također, u statističkom smislu procjene za parametre pojedinačnih modela bile manje pouzdane nego procjene za parametre modela koji u obzir uzima sve klase. Umjesto toga, bolje je raditi **multinomijalnu logističku regresiju** (MNR, nekad MLR), koju također zovemo i **klasifikator maksimalne entropije** (engl. *maximum entropy classifier*).

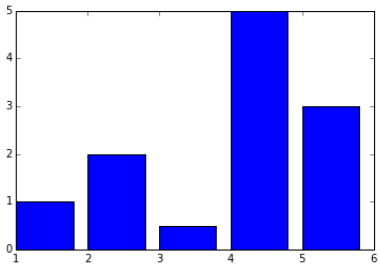
2.1 Model

Ideja je zapravo vrlo jednostavna: koristiti zaseban vektor težina $\mathbf{w}_k$ za svaku od K klasa, ali onda skalarni umnožak $\mathbf{w}_k^{\mathrm{T}}\mathbf{x}$ propustiti kroz prikladnu aktivacijsku funkciju koja će se pobrinuti da se vjerojatnosti svih klasa zbrajaju na ukupno 1. Funkcija koja radi baš to naziva se **funkcija softmax**. Za neki ulazni primjer $\mathbf{x}$, funkcija softmax uzima vrijednosti $\mathbf{w}_k^{\mathrm{T}}\mathbf{x}$ za svaku od K klasa, dakle K-dimenzijski vektor, te ih preslikava u K-dimenzijski vektor čije se komponente zbrajaju u 1. Formalno, softmax : $\mathbb{R}^n \to \mathbb{R}^n$, gdje je k-ta komponenta izlaznog vektora jednaka:

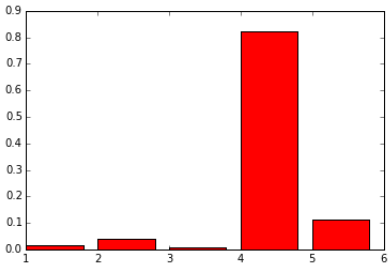
$$\mathrm{softmax}_k(x_1, \ldots, x_n) = \frac{\exp(x_k)}{\sum_j \exp(x_j)}$$

Funkcija softmax radi dvije stvari: **normalizira** sve vrijednosti tako da u zbroju budu 1, ali također **pojačava** veće vrijednosti i **smanjuje** manje vrijednosti. Funkcija se zove softmax jer odgovara funkciji max, ali je "meka" (u smislu da je, za razliku od funkcije max, neprekidna i diferencijabilna). Pogledajmo jedan primjer. 14

▸ **PRIMJER**



$\mathbf{x} = (1, 2, 0.5, 5, 3)$



$\mathrm{softmax}(\mathbf{x}) = (0.015, 0.041, 0.009, 0.823, 0.112)$

Model h multinomijalne logističke regresije definirat ćemo kao skup modela $\{h_k\}_k$, gdje je svaki model h_k zadužen za k-tu od ukupno K klasa. Svaki model h_k definirat ćemo tako da daje vjerojatnost da primjer $\mathbf{x}$ pripada klasi k, pomoću funkcije softmax: 15

$$h_k(\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_k^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))} = P(y = k|\mathbf{x}, \mathbf{W})$$

gdje je $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K)$ matrica sastavljena od K vektora težina $\mathbf{w}_k$. Primijetite da model h_k za klasu k kroz funkciju softmax uzima u obzir izlaze svih ostalih $(k-1)$ modela za preostale klase.

Time je definiran model. Sada ćemo izvesti funkciju pogreške.

2.2 Funkcija pogreške

Kod binarne logističke regresije funkciju pogreške izveli smo krenuvši od definicije negativnog logaritma vjerojatnosti oznaka. Oznake su bile binarne, $y \in \{0, 1\}$, odnosno to su bile **Bernoullijeve varijable**. Međutim, kako izlaz kod višeklasne regresije može poprimiti više od dvije vrijednosti ($K > 2$), prelazimo na **kategoričku varijablu**, koja se također naziva i **multinomijalna**, ili, možda bolje, **multinoullijeva** varijabla. Takvu varijablu prikazujemo kao **vektor indikatorskih (binarnih) varijabli**: 16

$$\mathbf{y} = (y_1, y_2, \ldots, y_K)^{\mathrm{T}}$$

gdje je $y_k = 1$, ako je ishod varijable k, a inače $y_k = 0$. Npr., $\mathbf{y} = (0, 0, 1, 0)$ označava da je multinomijalna varijabla poprimila treće stanje od četiri mogućih stanja. Pritom vrijedi $\sum_k y_k = 1$ (ishodi su međusobno isključivi i potpuni). Označimo vjerojatnost $P(y_k = 1)$ sa μ_k.

Sada ćemo definirati **distribuciju** te varijable. Prisjetimo se, distribucija Bernoullijeve varijable, koja ima samo dvije vrijednosti, $y = 0$ i $y = 1$, definirana je preko parametra μ na sljedeći način:

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

Ovo možemo poopćiti na $K \geqslant 2$ vrijednosti na sljedeći način. Najprije, treba nam K parametara, pa ćemo definirati vektor parametara:

$$\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$$

pri čemu za parametre μ_k vrijedi $\sum_k \mu_k = 1$ i $\mu_k \geqslant 0$, budući da predstavljaju vjerojatnosti.

Sada, po analogiji s Bernoullijevom distribucijom, distribuciju za kategoričku varijablu možemo definirati kao:

$$P(\mathbf{y}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{y_k}$$

Primijetimo: kao i kod binarne logističke regresije, **vjerojatnost da primjer $\mathbf{x}$ pripada klasi** k je upravo ono što nam daje model:

$$h_k(\mathbf{x}; \mathbf{W}) = \mu_k = P(y = k|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_k^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}$$

Sada konačno možemo napisati logaritam vjerojatnosti oznaka iz $\mathcal{D}$ kao:

$$\begin{aligned}\ln P(\mathbf{y}|\mathbf{X}) &= \ln \prod_{i=1}^{N} P(\mathbf{y}^i|\mathbf{x}) \\ &= \ln \prod_{i=1}^{N}\prod_{k=1}^{K} \mu_k^{y_k^i} = \ln \prod_{i=1}^{N}\prod_{k=1}^{K} h_k(\mathbf{x}^i; \mathbf{W})^{y_k^i} = \sum_{i=1}^{N}\sum_{k=1}^{K} y_k^i \ln h_k(\mathbf{x}^i; \mathbf{W})\end{aligned}$$

Funkcija pogreške koju želimo minimizirati je **negativan logaritam vjerojatnosti oznaka**:

$$E(\mathbf{W}|\mathcal{D}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^i \ln h_k(\mathbf{x}^i; \mathbf{W})$$

Vidimo da smo došli do **poopćenja pogreške unakrsne entroprije** na K klasa. Također, iz ovoga možemo iščitati da je funkcija gubitka jednaka:

$$L(\mathbf{y}, h_k(\mathbf{x})) = -\sum_{k=1}^{K} y_k^i \ln h_k(\mathbf{x}^i; \mathbf{W})$$

Logika je ista kao i kod binarne logističke regresije: ako je oznaka y_k^i nekog primjera i za klasu k jednaka 1, onda želimo da predikcija modela (izlaz softmaxa) bude visoka vjerojatnost blizu 1, jer će tada $\log h(\mathbf{x}) \approx 0$ i gubitak će biti nula. Inače, ako model za primjer čija je oznaka jednaka 1 daje vrijednost blizu 0, onda će logaritam biti velik negativan broj, njegova negacija bit će velik broj, pa će gubitak biti velik.

2.3 Optimizacija

Kao i kod binarne logističke regresije, ne možemo minimizirati $E(\mathbf{W}|\mathcal{D})$ u zatvorenoj formi, već trebamo raditi **iterativnu optimizaciju**. Za gradijentni spust, možemo se pokazati (doduše malo je nespretno) da je **gradijent funkcije pogreške** jednak: 17

$$\nabla_{\mathbf{w}_k} E(\mathbf{W}|\mathcal{D}) = \sum_{i=1}^{N} \big(h_k(\mathbf{x}^{(i)}; \mathbf{W}) - y_k^i\big)\boldsymbol{\phi}(\mathbf{x}^{(i)})$$

Ovo je gradijent po težinama posebno za klasu k. Ideja je da možemo ažurirati težine za svaku klasu posebno. Iz ovoga možemo direktno izvesti stohastički gradijentni spust (ažuriramo težine za svaki primjer, za svaku klasu). Možemo izvesti i standardni (grupni) gradijentni spust, gdje akumuliramo ažuriranja za sve primjere, za svaku klasu posebno. Možemo također izvesti i Newtonov postupak (Hesseovu matricu), ali to ćemo preskočiti.

2.4 "Online" učenje

Možda ste primijetili da je gradijent gubitka koju smo dobili za multinomijalnu logističku regresiju zapravo isti kao i onaj za binarnu logističku regresiju: **"pogreška modela puta vektor primjera"**. Kada to koristimo za učenje gradijentnim spustom, težine ažuriramo ovako:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\big(h(\mathbf{x}^{(i)}; \mathbf{w}) - y^i\big)\boldsymbol{\phi}(\mathbf{x}^{(i)})$$

Ovo pravilo smo već (u kontekstu perceptrona, ako se sjećate) rekli da se zove **Widrow-Hoffovo pravilo**, a drugi naziv je **least-mean-squares (LMS) algoritam** (ne brkati s least-squares, premda je očito povezano s time).

Algoritam LMS, odnosno ovakvo učenje gdje na temelju **stohastičkog gradijentnog spusta** minimiziramo pogrešku ažurirajući težine modela na gore definirani način, omogućava nam da radimo **online učenje**. O tome smo nešto rekli prošli puta. Podsjetimo se: online učenje je način učenja kod kojega ne moraju svi primjeri za učenje biti odmah dostupni, nego oni mogu dolaziti jedan po jedan, a težine modela će se ažuirati kako dolaze novi primjeri.

Vratimo se nakratko nekoliko tjedana unazad, na **linearnu regresiju**. Kada smo radili linearnu regresiju, pričali smo zapravo samo o **grupnoj optimizaciji**, koja se ostvaruje izračunom **pseudoinverza matrice dizajna**. Međutim, i tamo smo već mogli napraviti stohastičko (tj. online) ažuriranje težina. Naime, umjesto da analitički nalazimo minimum funkcije pogreške, možemo izračunati gradijent funkcije pogreške, pa primijeniti gradijentni spust. (Mi to tada nismo napravili, vjerojatno zato što smo bili zaslijepljeni čistom elegancijom rješenja u zatvorenoj formi). Pa, napravimo to sada. Funkcija kvadratne pogreške linearne regresije jest:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}^{(i)}) - y^i)^2$$

Gradijent (za jedan primjer $\mathbf{x}^{(i)}$) jest:

$$\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}|\mathcal{D}) = (\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}^{(i)}) - y^i)\boldsymbol{\phi}(\mathbf{x}^{(i)})$$

Dakle, pravilo za ažuriranje težina jest:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}^{(i)}) - y^i)\boldsymbol{\phi}(\mathbf{x}^{(i)})$$

što je opet pravilo LMS! Izgleda, dakle, da tu postoji neka zajedničkost – sva tri algoritma regresije za online učenje (stohastički gradijentni spust) koriste isto pravilo (LMS). Kako to? Pa, to je zato što svi ovi modeli koje smo radili – linearna regresija, logistička regresija i multinomijalna logistička regresija – pripadaju porodici **poopćenih linearnih modela**.

Sada je vrijeme da zaokružimo priču i damo jedan **unificirani pogled** na ta tri algoritma.

3 Poopćeni linearni modeli i eksponencijalna familija

Najprije, prisjetimo se s prošloga predavanja da su poopćeni modeli linearni modeli koji oko skalaranog umnoška vektora težina i vektora značajki imaju omotanu **aktivacijsku funkciju** f. Dakle:

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

Pogledajmo tri poopćena linearna modela kojima smo se bavili, i pogledajmo za svaki od njih četiri stvari: (1) kako je definiran **model**, (2) kojoj **vjerojatnosnoj distribuciji** odgovara njihov izlaz, (3) kako je definiran **gubitak** i (4) kako glasi **gradijent gubitka**, koji se koristi za gradijentni spust.

Prvo, pogledajmo algoritam **linearne regresije**:

$$\begin{aligned}
h(\mathbf{x};\mathbf{w}) &= f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) \\
P(y|\mathbf{x},\mathbf{w}) &= \mathcal{N}(\mu,\sigma^2) = \mathcal{N}(h(\mathbf{x}),\sigma^2) \\
L(y,h(\mathbf{x})) &= (h(\mathbf{x})-y)^2 \\
\nabla_{\mathbf{w}} L\big(y,h(\mathbf{x})\big) &= \big(h(\mathbf{x})-y\big)\boldsymbol{\phi}(\mathbf{x})
\end{aligned}$$

Za grupno učenje koristimo postupak najmanjih kvadrata (pseudoinverz), a za online učenje koristimo pravilo LMS.

Pogledajmo algoritam **logističke regresije**:

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

Za grupno učenje koristimo gradijentni spust, Newtonov postupak (IRLS) ili kvazi-Newtonov postupak (BFSG, L-BFSG). Za online učenje koristimo pravilo LMS.

Konačno, pogledajmo **multinomijalnu logističku regresiju**:

$$\begin{aligned}
h_k(\mathbf{x};\mathbf{W}) &= \mathrm{softmax}(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})) = \frac{\exp(\mathbf{w}_k^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))} = P(y=k|\mathbf{x},\mathbf{w}) \\
P(\mathbf{y}|\mathbf{x},\mathbf{w}) &= \prod_{k=1}^{K}\mu_k^{y_k} = \prod_{k=1}^{K} h_k(\mathbf{x})^{y_k} \\
L(\mathbf{y},h_k(\mathbf{x})) &= -\sum_{k=1}^{K} y_k \ln h_k(\mathbf{x};\mathbf{W}) \\
\nabla_{\mathbf{w}_k} L\big(y_k,h_k(\mathbf{x})\big) &= \big(h_k(\mathbf{x})-y_k\big)\boldsymbol{\phi}(\mathbf{x})
\end{aligned}$$

Za učenje modela koristimo iste (mutatis mutandis) optimizacijske postupke kao i za logističku regresiju.

Uočimo zajedničkosti. Kod sva tri algoritma funkciju gubitka izveli smo iz izraza za negativan logaritam vjerojatnosti oznaka primjera iz skupa primjera. Pritom smo za linearnu, binarnu logističku i multinomijalnu logističku regresiju koristili normalnu, Bernoullijevu odnosno multinoullijevu razdiobu. Nadalje, za sva tri algoritma izveli smo identično pravilo (LMS) za online ažuriranje težina.

Pitanje je: kako to da smo dobili uvijek isto pravilo ažuiranja težina? Također, koja je veza između logističke funkcije i Bernoullijeve varijable, te između funkcije softmax i multinoullijeve razdiobe? Čini se da je to povezano, jer smo u oba slučaja dobili pogrešku unakrsne entropije. Odgovor leži u svojstvima distribucija koje smo koristili za modeliranje oznaka $y^{(i)}$.

3.1 Eksponencijalna familija

Distribucije koje smo do sada susreli (Gaussova, Bernoullijeva, multinoullijeva), ali i neke druge koje se često koriste u strojnom učenju (binomna, multinomijalna, Studentova t-distribucija, uniformna, beta-distribucija, gamma-distribucija, Dirichletova) pripadaju **eksponencijalnoj familiji** (engl. *exponential family*). Što je eksponencijalna familija? Eksponencijalna familija je jedna široka grupa distribucija koje se mogu napisati u ovakvom obliku:

$$p(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp\big(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) - A(\boldsymbol{\theta})\big)$$

Eksponencijalna familija distribucija ima mnoga svojstva koja su važna za strojno učenje, ali uglavnom više za probabilističke pristupe, pa sada ovdje nećemo uopće dalje o tome.

Ono što nam je ovdje zanimljivo uočiti jest da je eksponencijalna familija ključna za poopćene linearne modele. Konkreto, za distribucije koje pripadaju eksponencijalnoj familiji (uključivo Gaussova, Bernoullijeva i multinoullijeva), postoji veza između distribucije i njezine (moguće nelinearne) aktivacijske funkcije f. Ta se funkcija u tom kontekstu naziva **funkcija sredine** (engl. *mean function*), zato što definira parametar μ distribucije, a to je parametar srednje vrijednosti distribucije. Aktivacijska funkcija f dakle definira μ kao funkciju od $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$. Pogađate, za Gaussovu distribuciju aktivacijska funkcija je funkcija identiteta, budući da $\mu = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$. Za Bernoullijevu distribuciju to je logistička funkcija, a za multinoullijevu distribuciju to je funkcija softmax. 18

Nakon ove inspirativne teme, pogledajmo još jednu ne manje inspirativnu stvar...

4 Adaptivne bazne funkcije

Kod poopćenih linearnih modela (i za regresiju i za klasifikaciju) imali smo mogućnost primjere preslikati u prostor značajki pomoću **funkcije preslikavanja značajki**:

$$\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^{m+1}$$
$$\boldsymbol{\phi}(\mathbf{x}) = \big(1, \phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})\big)$$

gdje je $\{\phi_1, \phi_2, \ldots, \phi_m\}$ skup od m **baznih funkcija** (nelinearnih funkcija ulaznih varijabli): $\phi_j : \mathbb{R}^n \to \mathbb{R}$. Npr., polinomijalno preslikavanje za $n = 2$ i $d = 2$:

$$\boldsymbol{\phi}(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

Takvo preslikavanje smo onda vrlo lako ugradili u bilo koji poopćeni linearni model:

$$h(\mathbf{x}; \mathbf{w}) = f\big(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\big) = f\Big(\sum_{j=0}^{m} w_j\phi_j(\mathbf{x})\Big)$$

gdje je f neka odabrana aktivacijska funkcija (odnosno funkcija sredine, da odmah iskoristimo netom naučen pojam).

Premda mi to nismo isprobavali, bazne funkcije ϕ_j ne moraju nužno biti potencije ili faktori ulaznih značajki, već to mogu biti raznorazne funkcije. Jedna zanimljiva mogućnost su funkcije koje mjere sličnost primjera s nekim prototipnim primjerima u ulaznome prostoru. To se onda zove **jezgreni stroj**, i o tome ćemo pričati za dva tjedna.

Kako god, ograničavajuće je to što su ovo **fiksne bazne funkcije**: njihov broj i oblik je unaprijed određen. To je problem zato što mi u većini slučajeva ne znamo unaprijed koje su bazne funkcije dobre za naš problem. Drugim riječima, općenito ne znamo koje preslikavanje u prostor značajki će naš problem u tom prostoru učiniti linearno odvojivim.

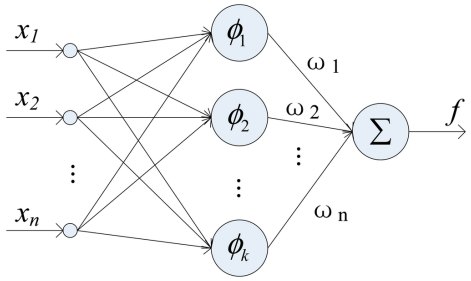
Problem možemo riješiti tako da bazne funkcije **prilagodimo** našim podatcima (primjerima za učenje). Tu imamo dvije mogućnosti. Prva mogućnost, koju koriste spomenuti jezgreni strojevi, jest da odaberemo neke primjere iz skupa za učenje kao prototipe, a onda bazne funkcije mjere sličnost ulaznog primjera s tim prototipima. Tu se ukupan broj baznih funkcija prilagođava podatcima. Druga mogućnost je da ipak koristimo fiksni broj baznih funkcija, ali da dozvolimo da se svaka od njih prilagodi podatcima. Pogledajmo to malo detaljnije.

Ideja prilagodivih baznih funkcija jest da ih definiramo do na neke parametre, koje onda možemo prilagoditi na podatcima. Dakle, imat ćemo **parametrizirane bazne funkcije**. Zvuči li to poznato? Dakako. To je upravo kako učimo modele u strojnom učenju: definiramo funkciju do na neke parametre, a parametre odredimo optimizacijom empirijske pogreške na skupu za učenje. No, pogledajmo prvo kako bismo parametrizirali bazne funkcije. Jedna mogućnost jest da kažemo da je svaka bazna funkcije jedan malen poopćeni linearni model za sebe! Dakle, imat ćemo poopćeni linearni model i unutar njega imat ćemo poopćene linearne modele kao njegove bazne funkcije:

$$h(\mathbf{x};\mathbf{w}) = f\Big(\sum_{j=0}^{m} w_j^{(2)} \underbrace{f\big(\sum_{i=0}^{n} w_{ji}^{(1)} x_i\big)}_{=\phi_j(\mathbf{x})}\Big) = f\big(\mathbf{w}^{(2)\mathrm{T}} f(\mathbf{W}^{(1)}\mathbf{x})\big)$$

Primijetite da svaka bazna funkcija našeg modela treba imati svoj vektor težina, stoga težine w_{ji} u unutarnjoj sumi imaju dva indeksa: j je indeks bazne funkcije ϕ_j, a i je indeks težine u vektoru težina bazne funkcije ϕ_j. Sa superskriptom $^{(1)}$ odnosno $^{(2)}$ označili smo koje se težine prve koriste u izračunu predikcije modela, a koje se koriste druge. Izraz na desnoj strani je samo matrični zapis modela, gdje smo se uspjeli rješiti suma i težine smo objedinili u vektor težina i matricu težina. Uzmite si malo vremena da vidite da je taj zapis ekvivalentan zapisu sa sumama.

Sada kada smo ovo metabolizirali, vrijeme je za veliko otkrivenje. Kakav je zapravo ovaj model? Za svaku baznu funkciju ϕ_j imamo težine iz matrice $\mathbf{W}^{(1)}$, koje množimo sa svim značakama ulaznog vektora i zbrajamo. Zatim te vrijednosti opet množimo sa težinama $\mathbf{w}^{(2)}$ i zbrajamo. Dobili smo nešto što je većini vas već poznato: **neuronsku mrežu**!



Ova naša mreža je dvoslojna, međutim ništa nas ne sprječava da idemo dublje: da su bazne funkcije i same kombinacije drugih baznih funkcija, a one opet kombinacija trećih baznih funkcija, itd.

Očito, neuronske mreže su složeniji model od poopćenih linearnih modela. To, naravno, ima i svoju cijenu: složeniji postupak optimizacije (zbog **nekonveksnosti funkcije pogreške**, budući da gubitak u izlaznom sloju – koji i dalje može biti kvadratni gubitak ili gubitak unakrsne entropije – sada ima vrlo složenu ovisnost o težinama prethodnih slojeva) te veća mogućnost **pretreniranosti modela**. Naravno, za sve to postoje razni prijedlozi rješenja, posebice u okviru

danas popularne paradigme dubokog učenja. Nećemo ići dalje, međutim, budući da se ova tema poprilično detaljno pokriva u drugim predmetima.

Nama je ovdje samo bitno da uočite poveznicu: neuronska mreža je proširenje poopćenog linearnog modela kod kojega su bazne funkcije prilagodive, odnosno funkcija preslikavanja značajki također se uči iz podataka.

Sažetak

- **Newtonov postupak** je optimizacija drugog reda koja konvergira brže od gradijentnog spusta, a temelji se na izračunu **Hesseove matrice**. Varijanta za logističku regresiju zove se **IRLS**
- Izračun Hesseove matrice je vremenski i prostorno skup, pa možemo koristiti kvazi-Newtonov postupak, kao što je **L-BSFG**
- **Multinomijalna logistička regresija** je poopćenje logističke regresije na više od dvije klase, sa **softmax** funkcijom kao aktivacijskom funkcijom
- Zajedničko **poopćenim linearnim modelima** jest da su njihovi izlazi varijable iz **eksponencijalne familije** distribucija
- Umjesto fiksnih, možemo koristiti parametrizirane **adaptivne bazne funkcije**, što nas dovodi do **neuronskih mreža**

Bilješke

[1] **Newtonov optimizacijski postupak** naziva se nekad i **Newton-Raphsonov postupak**, premda to nije skroz točno. Newton-Raphsonov postupake je iterativan postupak za nalaženje nultočaka funkcije. Newtonov optimizacijski postupak koristi Newton-Raphsonov postupak za nalaženje nultočke prve derivacije funkcije koja se optimizira. Dakle, Newtonov postupak optimizacije koristi Newton-Raphsonov postupak. Inače, Newton-Raphsonov postupak osmislio je Isaac Newton 1685. godine, a Joseph Raphson, također engleski matematičar, predložio je sličnu metodu 1690. godine u ponešto simplificiranoj varijanti, i s nekim referencama na Newtonov rad. Postupak, dakle, nisu razvili zajednički, a čini se i da su slabo komunicirali (to možda ne čudi, jer je Newton navodno bio teška i agresivna osoba). O povijesti razvoja Newton-Raphsonovog postupka možete pročitati u (Ypma, 1995). Usput, osim što je poznat po Newton-Raphsonovom postupku, Joseph Raphson poznat je i po tome što je prvi uveo naziv "panteizam" za filozofski pravac koji je utemeljio prosvjetiteljski filozof Baruch Spinoza (pojednostavljeno: Bog=priroda), a dvjesto godina kasnije tematizirao vrlo utjecajan američki filozof i književnik Ralph Waldo Emerson.

[2] Ovdje se prirodno nameće pitanje: zašto radimo aproksimaciju samo drugog reda? Zašto ne trećeg ili višeg reda? Ne bi li to optimizaciju učinilo još učinkovitijom? Odgovor je: ne bi. Kvadratna aproksimacija je dovoljno dobra, u smislu da nije računalno presložena, a konvergira brže od optimizacije prvog reda. V. ovdje: https://stats.stackexchange.com/q/320082/93766.

[3] U literaturi na hrvatskome jeziku ponekad se, čini mi se pogrešno, koristi naziv "Hessova matrica". Naime, **Hesseovu matricu** uveo je njemački matematičar Ludwig Otto Hesse (a ne "Hess"!), rođen u Königsbergu u tadašnoj Prusiji, a današnjem Kalinjingradu, i to nekoliko godina nakon smrti Immanuela Kanta, koji je također živio u Königsbergu, kojega, navodno, nikada nije napustio. Ferovcima će možda biti zanimljivo znati da je Hesse bio doktorskim mentorom Gustavu Kirchoffu, koji je osmislio poznate Kirchoffove zakone, i to dok je još bio student. Konačno, napomenimo, za svaki slučaj, da Ludwig Hessse i Hesseova matrica nemaju nikakve veze s književnikom Hermannom Hesseom, rođenom nakon smrti Ludwiga Hessea (osim ako ne postoji neka veza između druge derivacije vektorske funkcije i spiritualnog samootkrivenja koje Hesse opisuje u Siddharthi).

[4] **Hesseovu matricu** nemojte brkati s **Jakobijevom matricom** ("Jakobijanom"). Za razliku od Hesseove matrice, koja je matrica parcijalnih derivacija **drugog reda** funkcije $f : \mathbb{R}^n \to \mathbb{R}$, tj. vektorske funkcije

koja vraća skalar, Jakobijeva matrica je matrica parcijalnih derivacija **prvog reda** vektorske funkcije koja vraća vektor, $f : \mathbb{R}^n \to \mathbb{R}^m$:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

gdje je f_i i-ta komponenta izlaza funkcije f. Drugim riječima, element Jakobijeve matrice je:

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$$

Zanimljivo, Jakobijanu je uveo Carl Gustav Jacob Jacobi, koji je bio doktorski mentor upravo Ludwigu Hesseu. Eto kako je svijet malen.
Premda se i Hesseova matrica $\mathbf{H}$ i Jakobijeva matrica $\mathbf{J}$ redovito se koriste u strojnom učenju, mi ćemo na ovom predmetu trebati samo Hesseovu (Jakobijeva će vam trebati za izračun gradijenata kod neuronskih mreža u dubokom učenju).

[5] Izvod za minimum od f_{quad} možete naći u (Murphy, 2012) (dio 8.3.3).

[6] Dokaz pogledajte ovdje: https://math.stackexchange.com/q/2083629/273854.

[7] Vidi (Bishop, 2006) (dio 4.3.3) ili (Murphy, 2012) (dio 8.3.1).

[8] Dokaz pogledajte ovdje: https://math.stackexchange.com/q/3189729/273854

[9] Izvod **algoritma najmanjih kvadrata s iterativnim ažuriranjem težina (IRLS)** možete naći u (Bishop, 2006) (dio 4.3.3) i (Murphy, 2012) (dio 8.3.4). Naziv algoritma reflektira činjenicu da pravilo za ažuriranje težina (nismo ga napisali; v. citiranu literaturu) nalikuje jednadžbi za izračun težina Moore-Penroseovim pseudoinverzom, kakvu smo imali kod optimizacijskog postupka najmanjih kvadrata kod linearnog modela regresije, uz tu razliku da je izračun potrebno obavljati iterativno.

[10] Algoritam **BFSG** nazvan je prema matematičarima Broydenu, Fletcheru, Goldfarbu i Shannou, koji su ga nezavisno osmilili 1970. godine. Čini se da poredak imena ne odgovara kronološkom slijedu objavljivanja članaka, pa je vjerojatno dogovorno uzet abecedan poredak. Kako god, lijepo je da su spomenuti svi izumitelji. Nažalost, rijetki su te sreće; v. **Stiglerov zakon eponimije**: https://en.wikipedia.org/wiki/Stigler%27s_law_of_eponymy.

[11] Metode **aproksimacije matricom nižeg ranga** (engl. *low-rank matrix approximations*) intenzivno se koriste u nekim dijelovima strojnog učenja, pogotovo u tehikama za smanjenje dimenzionalnosti i u jezgrenim metodama. Začudo, nama u ovom predmetu te metode neće trebati. Ako želite pročitati nešto o tim metodama, predlažem krenuti od (Kishore Kumar and Schneider, 2017).

[12] Algoritam L-BFSG osmislio je američki matematičar i računarac Jorge Nocedal. Osnovno o algoritmu L-BFSG možete pročitati na https://en.wikipedia.org/wiki/Limited-memory_BFGS. Detaljniji opis možete naći 9. poglavlju Nocedalove knjige (Nocedal and Wright, 2006). Ovo je također dobar opis: https://aria42.com/blog/2014/12/understanding-lbfgs, unatoč snipetima Java koda. Na dotičnoj stranici možete naći i fotografiju na kojoj kvartet BFSG pije pivu. Nocedal je algoritam L-BFSG izvorno napisao u Fortranu; ako želite vidjeti kako taj kod izgleda, pogledajte ga ovdje: http://users.iems.northwestern.edu/~nocedal/lbfgs.html

[13] **Podgradijent** (engl. *subgradient*) je iterativan optimizacijski postupak za minimizaciju nediferencijabilne konveksne funkcije. U strojnom učenju tipično se koristi za minimizaciju L_1-regulariziranih pogrešaka ili pogrešaka izvedenih iz nediferencijabilne funkcije gubitka. Detaljnije u (Boyd et al., 2003).

[14] Zapravo, funkcija softmax prije "mekoj" verziji funkcije argmax nego max. Naime, funkciju argmax možemo interpretirati kao funkciju koja na izlazu daje tzv. **one-hot vektor** – vektor binarnih indikacijskih varijabli koje su sve jednake nula osim jedne, koja je jednaka jedinici. Ta jedna jedinica odgovara indeksu ulaznog vektora na kojemu se nalazi maksimalna vrijednost. Npr., $\text{argmax}(1, 4, 2, 3) =$

$(0, 1, 0, 0)$. Funkcija softmax onda se može shvatiti kao "meka" varijanta funkcije argmax jer na izlazu daje "zaglađeni" one-hot vektor. Npr., $\text{softmax}(1, 4, 3, 2) = (0.03, 0.64, 0.24, 0.09)$. Više ovdje: https://medium.com/@u39kun/is-the-term-softmax-driving-you-nuts-ee232ab4f6bd.

15 Ovdje se možda da naslutiti da je aktivacijska funkcija softmax zapravo poopćenje sigmoidne funkcije na više od dvije klase. Degenerirani slučaj funkcije softmax je slučaj kada $K = 2$, što će biti isto kao i izlaz sigmoidne funkcije, ali sa dvodimenzijskim vektorom kao izlazom: $(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}), 1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}))$. V. https://stats.stackexchange.com/a/254071/93766.

16 Ne stižem ovdje napisati propisnu bilješku o nazivu **multinoullijeva varijabla**. Kao privremenu kompenzaciju, nudim ovaj tekst: https://geekyisawesome.blogspot.com/2016/12/bernoulli-vs-binomial-vs-multinoulli-vs.html. Naziv "multinullijeva varijabla" (kao stopljenicu od "Bernoullijeva" i "multinomijalna") predložio je Murphy u (Murphy, 2012), ali nisam siguran da se uhvatila. Najsigurnije je koristiti "kategorička varijabla". Naziv "multinomijalna varijabla" je u ovom kontekstu, zapravo, pogrešan.

17 Za izvod, pogledati https://math.stackexchange.com/q/2852620/273854

18 Inverz funkcije sredine naziva se **vezna funkcija** (engl. *link function*). Za funkciju identiteta to je funkcija identiteta, a za logističku/softmax funkciju to je funkcija **logit**. Kolokvijalno, "logiti" su vrijednosti koje ulaze u softmax funkciju (dakle vrijednost $\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x})$ kod logističke regresije, odnosno vrijednosti $\mathbf{w}_k^{\mathrm{T}}\phi(\mathbf{x})$ za svaku klasu kod multinomijalne regresije).

Literatura

C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

S. Boyd, L. Xiao, and A. Mutapcic. Subgradient methods. *lecture notes of EE392o, Stanford University, Autumn Quarter*, 2004:2004–2005, 2003.

N. Kishore Kumar and J. Schneider. Literature survey on low rank approximation of matrices. *Linear and Multilinear Algebra*, 65(11):2212–2244, 2017.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

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

T. J. Ypma. Historical development of the Newton–Raphson method. *SIAM review*, 37(4): 531–551, 1995.