

## 7. Logistička regresija II

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

Jan Šnajder, predavanja, v1.8

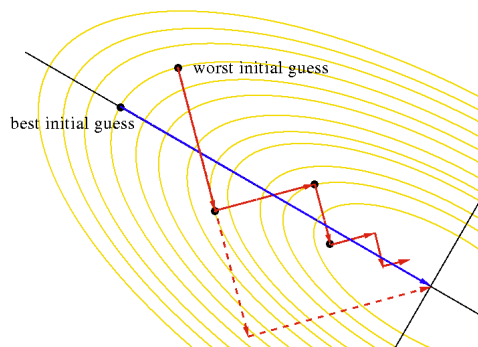
Prošli put upoznali smo se s algoritmom **logističke regresije**. Definirali smo model i zatim smo izveli funkciju pogreške unakrsne entropije kao negativan logaritam vjerojatnosti oznaka iz skupa za učenje. Ustanovili smo da minimizacija te pogreške nema rješenje u zatvorenoj formi, pa smo se okrenuli 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 **višeklasno proširenje logističke regresije**. Treće, malo ćemo se osvrnuti na dosada naučeno i pogledati zajedničkosti poopćenih linearnih modela. 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 trebati.

### 1 Alternative gradijentnom spustu

Prošli smo put ustanovili da gradijentni spust treba linijsko pretraživanje, jer u suprotnom nemamo zajamčenu **globalnu konvergenciju**, što znači da se, ovisno o početnoj točki pretraživanja, može dogoditi da gradijentni spust ne konvergira već da divergira (efektivno se umjesto spuštanja počinje podizati). 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 vrlo 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 nalazi 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 svaki korak radimo na temelju gradijenta izračunatog za jedan pojedinačni primjer. U nastavku se fokusiramo na optimizaciju koja nije stohastička, dakle optimizaciju koja se temelji na grupnom gradijentnom spustu.

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 gradijentnog spusta koji je optimizacijski postupak prvog reda. Osnovni postupak drugog reda jest **Newtonov postupak**.

1

## 1.1 Newtonov postupak

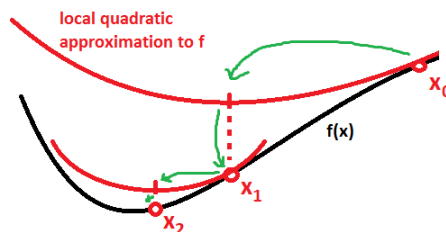
Razmotrimo minimizaciju funkcije  $f(\mathbf{x})$ . Ažuriranje parametara kod gradijentnog spusta bilo je:

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

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

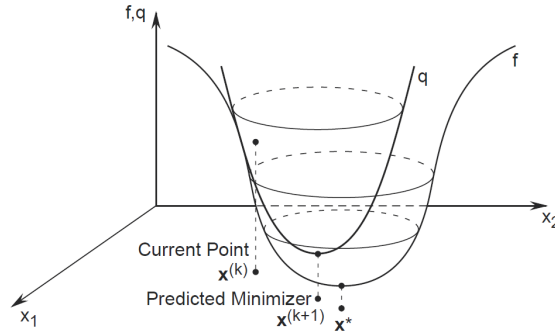
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

Ideja kod Newtonovog algoritma jest da u točki  $\mathbf{x}_t$  (trenutačni minimum) napravimo kvadratnu aproksimaciju funkcije  $f(\mathbf{x})$ . 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.

Postupak funkcionira identično za funkciju više varijabli, tj. u višedimenzijaskome prostoru parametara. U dvodimenzijaskome 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 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 takozvani **Taylorov red 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:

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

gdje je  $\mathbf{H}_t$  **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  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ , tj. funkcije koja  $n$ -dimenzijski vektor preslikava u skalar. Definirana je ovako: 3

$$\mathbf{H} = \nabla \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

to jest, element Hesseove matrice definiran je s: 4

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

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

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

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

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**. Općenito, Hesseova matrica  $\mathbf{H}$  je **pozitivno semi-definitna** ( $\mathbf{x}^T \mathbf{H} \mathbf{x} \geq 0$  za svaki ne-nul vektor  $\mathbf{x}$ ) ako i samo ako je funkcija  $f(\mathbf{x})$  **konveksna**. Pogrešna unakrsne entropije je konveksna, pa će dakle matrica  $\mathbf{H}$  za logističku regresiju biti pozitivno semi-definitna. Međutim, inverz postoji samo za pozitivno definitne matrice ( $\mathbf{x}^T \mathbf{H} \mathbf{x} > 0$  za svaki ne-nul vektor  $\mathbf{x}$ ). Ako je  $\mathbf{H}$  pozitivno-semidefinitna, a nije ujedno i pozitivno-definitna, onda ne postoji inverz od  $\mathbf{H}$  i ne možemo primijeniti Newtonov postupak. To zapravo znači da postoji linearna zavisnost među značajkama, pa treba ili odabrati podskup značajki ili napraviti regularizaciju. Na sreću, kod logističke regresije matrica  $\mathbf{H}$  je pozitivno definitna, pa dakle uvijek ima inverz. No, na žalost, zbog multikolinearnosti, matrica može biti nekondicionirana i rješenje će biti nestabilno, što znači da opet trebamo ili odabrati podskup značajki ili provesti regularizaciju (vidjet ćemo uskoro kako). 6  
7

## 1.2 Newtonov postupka za logističku regresiju

Primijenimo sada Newtonov postupak na logističku regresiju. Lako se može izvesti da je Hesseova matrica za pogrešku unakrsne entropije sljedeća: 8

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

gdje je  $\mathbf{S} = \text{diag}(h(\mathbf{x}^{(i)})(1 - h(\mathbf{x}^{(i)})))$ , tj. dijagonalna matrica koja na dijagonali ima vrijednosti derivacije 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  $\mathbf{H}$  je  $(m + 1) \times (m + 1)$ , gdje je  $m$  broj značajki u prostoru značajki (dakle, nakon preslikavanja).

Ako bismo sada uvrstili konkretan izraz za  $\mathbf{H}$  u maloprije spomenuto pravilo za ažuriranje parametara:

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

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 i njezinog 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 trenutnog i prethodnog koraka). Najpoznatija takva metoda jest algoritam **BFGS**. 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**. Taj algoritam zove se **ograničen BFSG** (engl. *limited BFSG*, **L-BFSG**). Opet, dovoljno je samo da znate da to postoji i što to je.

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 linearno odvojivih problema imamo  $\|\mathbf{w}\| \rightarrow \infty$  i logistička se regresija lako prenaučí). 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  $\lambda$  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 modela linearne regresije. Sve u svemu,  $L_2$ -regularizacija je jedna pitoma zvjerká.

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 metode** optimizacijske postupke. Za sada ne trebate znati ništa više osim da je ovo moguće, da postoje algoritmi i da su implementirani u alatima koje koristite.

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:

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

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, a također bi 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

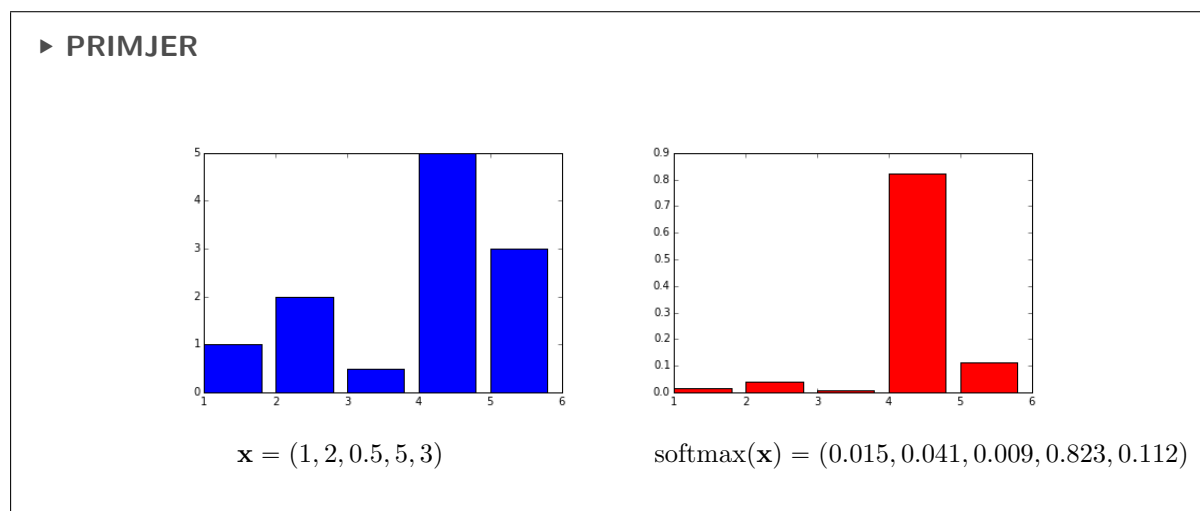
Ideja je zapravo vrlo jednostavna: koristiti zaseban vektor težina  $\mathbf{w}_k$  za svaku od  $K$  klasa, ali onda skalarni umnožak  $\mathbf{w}_k^T \mathbf{x}$  propustiti kroz prikladnu aktivacijsku funkciju koja će se pobrinuti da se vjerojatnosti svih klasa zbrajaju na ukupno 1. Funkcija koja radi baš to naziva se **funkcija softmax**. Za neki ulazni primjer  $\mathbf{x}$ , funkcija softmax uzima vrijednosti  $\mathbf{w}_k^T \mathbf{x}$  za svaku od  $K$  klasa,

dakle  $K$ -dimenzijski vektor, te ih preslikava u  $K$ -dimenzijski vektor čije se komponente zbrajaju u 1. Formalno,  $\text{softmax} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , gdje je  $k$ -ta komponenta izlaznog vektora jednaka:

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

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

14



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

15

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

gdje je  $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K)$  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 definirali smo negativnan logaritam 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 ( $K > 2$ ) vrijednosti, onda prelazimo na **kategoričku varijablu**, koja se također zove i **multinomijalna**, ali bolje **multinoullijeva** varijabla. Takvu varijablu prikazujemo kao **vektor indikatorskih (binarnih) varijabli**:

16

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

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  $\mu_k$ .

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  $\mu$  na sljedeći način:

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

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

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

pri čemu za parametre  $\mu_k$  vrijedi  $\sum_k \mu_k = 1$  i  $\mu_k \geq 0$ , 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:

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

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

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

**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 je ovo **poopćenje pogreške unakrsne entropije** na  $K$  klasa. Također, iz ovoga možemo iščitati da je funkcija gubitka jednaka:

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

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 pogreška će biti nula. Inače, ako model za primjer čija je oznaka jednaka 1 daje vrijednost blizu 0, onda će logaritam biti velik negativan broj, koji će se množiti s 1, pa će gubitak biti velik (minus puta minus).

### 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:

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

Ovo je gradijent po težinama za jednu 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), a 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, a to je **pogreška modela puta vektor primjera**. Kada to koristimo za učenje gradijentnim spustom, težine ažuriramo ovako:

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

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 na ovaj način težine, nam omogućava da radimo **online učenje**. O tome smo nešto rekli prošli puta. Podsjetimo se: online učenje je učenje kod kojega ne moraju svi primjeri za učenje biti odmah dostupni, nego oni mogu dolaziti jedan po jedan, a mi ćemo na svakom primjeru trenirati model.

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 mogli napraviti on (tj. pojedinačno) ažuriranje težina. Naime, umjesto da tražimo analitički minimum funkcije pogreške, možemo izračunati gradijent funkcije pogreške, pa primijeniti gradijentni spust (mi to nismo napravili nismo jer smo bili zaslijepljeni elegancijom rješenja u zatvorenoj formi). Pa, napravimo to sada. Funkcija kvadratne pogreške linearne regresije jest:

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

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

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

Dakle, pravilo za ažuriranje težina jest:

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

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

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

## 3 Poopćeni linearni modeli i eksponencijalna familija

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**:

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

$$P(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mu, \sigma^2) = \mathcal{N}(h(\mathbf{x}), \sigma^2)$$

$$L(y, h(\mathbf{x})) = (h(\mathbf{x}) - y)^2$$

$$\nabla_{\mathbf{w}} L(y, h(\mathbf{x})) = (h(\mathbf{x}) - y) \phi(\mathbf{x})$$



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

Pogledajmo algoritam **logističke regresije**:

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

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

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

$$\nabla_{\mathbf{w}} L(y, h(\mathbf{x})) = (h(\mathbf{x}) - y) \phi(\mathbf{x})$$

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**:

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

$$P(\mathbf{y} | \mathbf{x}, \mathbf{w}) = \prod_{k=1}^K \mu_k^{y_k} = \prod_{k=1}^K h_k(\mathbf{x})^{y_k}$$

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

$$\nabla_{\mathbf{w}_k} L(y_k, h_k(\mathbf{x})) = (h_k(\mathbf{x}) - y_k) \phi(\mathbf{x})$$

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 oznaka iz skupa primjera. Pritom smo za linearnu, logističku i multinomijalnu logističku koristili normalnu, Bernoullijevu odnosno multinoullijevu razdiobu. Nadalje, za sva tri algoritma izveli smo identično pravilo za online ažuriranje težina (LMS).

Pitanje je: kako to da smo dobili uvijek isto pravilo ažuriranja 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  $y$ .

### 3.1 Eksponecijalna 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 **eksponecijalnoj familiji** (engl. *exponential family*). Što je eksponecijalna familija? Eksponecijalna familija je jedna široka grupa distribucija koje se mogu napisati u ovakvom obliku:

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

Eksponecijalna 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 eksponecijalna familija **ključna za poopćene linearne modele**. Konkreto, za distribucije koje pripadaju eksponecijalnoj 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  $\mu$  distribucije, a to je parametar srednje vrijednosti distribucije. Aktivacijska funkcija  $f$  dakle definira  $\mu$  kao funkciju od  $\mathbf{w}^T \phi(\mathbf{x})$ . Pogađate, za Gaussovu distribuciju aktivacijska funkcija je funkcija identiteta, budući da  $\mu = \mathbf{w}^T \phi(\mathbf{x})$ . Za Bernoullijevu distribuciju to je logistička funkcija, a za multinoullijevu distribuciju to je funkcija softmax.

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**:

$$\begin{aligned}\phi &: \mathbb{R}^n \rightarrow \mathbb{R}^{m+1} \\ \phi(\mathbf{x}) &= (1, \phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))\end{aligned}$$

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

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

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

$$h(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x})) = f\left(\sum_{j=0}^m w_j \phi_j(\mathbf{x})\right)$$

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

Premda mi to nismo isprobavali, treba znati da bazne funkcije  $\phi_j$  ne moraju nužno biti potencije ili faktori ulaznih značajki, već mogu biti različite funkcije, npr., 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, može biti problematično 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 odvojjivim.

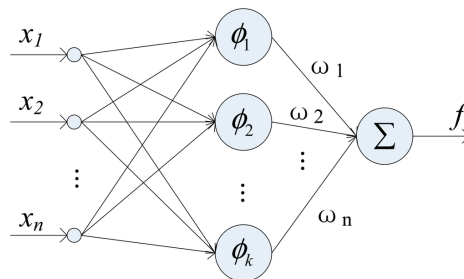
Problem možemo riješiti tako da bazne funkcije prilagodimo našim podatcima (primjerima za učenje). Tu imamo dvije mogućnosti. Prva mogućnost, koju koriste spomenuti jezgreni strojevi, jest da odaberemo neke primjere iz skupa za učenje kao prototipe, a onda bazne funkcije mjere sličnost ulaznog primjera s tim prototipima. Tu se skup (i ukupan broj) baznih funkcija prilagođava podatcima. Druga mogućnost je da ipak koristimo fiksni broj baznih funkcija, ali da dozvolimo njihovo prilagođavanje 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 poopćene linearne modele kao bazne funkcije:

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

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  $\phi_j$ , a  $i$  je indeks težine u vektoru težina parametrizirane bazne funkcije  $\phi_j$ . Sa superskriptom <sup>(1)</sup> odnosno <sup>(2)</sup> označili smo koje se težine prve koriste u izračunu predikcije modela, a koje se koriste druge. Izraz na desnoj strani je samo matricni zapis modela, gdje smo se uspjeti riješiti suma i težine smo objedinili u vektor težina odnosno u 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. Što je ovaj model zapravo? Za svaku baznu funkciju  $\phi_j$  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, ali to naravno ima i svoju cijenu: složeniji postupak optimizacije (**nekonveksnost funkcije pogreške**, jer 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 **pretriranosti modela**, osobito kod dubokih mreža. Naravno, za sve to postoje razna rješenja ili prijedlozi rješenja, posebice u okviru danas popularne paradigme dubokog učenja. Ali o tome nećemo dalje u ovom predmetu, jer se ova tema vrlo detaljno obrađuje u mnogim 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 se također 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, npr. **L-BSFG**
- **Multinomijalna logistička regresija** je poopćenje logističke regresije na više klasa, sa **softmax** funkcijom kao izlazom
- Zajedničko **poopćenim linearnim modelima** jest da su njihovi izlazi varijable iz **eksponencijalne familije** distribucija
- Umjesto fiksnih, možemo koristiti parametrizirane **adaptivne bazne funkcije**, što nas doводи do **neuronskih mreža**

## Bilješke

- 1 **Newtonov optimizacijski postupak** naziva se nekad i **Newton-Raphsonov postupak**, premda to nije skroz točno. Newton-Raphsonov postupak 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 Rapson 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 tadašnjoj Prusiji, a današnjem Kalinjingradu, i to nekoliko godina nakon smrti Immanuel 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 Hesse i Hesseova matrica nemaju nikakve veze s književnikom Hermannom Hesseom, rođenom nakon smrti Ludwiga Hesea (premda ostavljam ostvorenom mogućnost postojanja veze 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  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , tj. vektorske funkcije koja vraća skalar, Jakobijeva matrica je matrica parcijalnih derivacija **prvog reda** vektorske funkcije koja vraća vektor,  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ :

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

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  $f_{\text{quad}}$  možete naći u [Murphy, 2012] (dio 8.3.3).
- 6 Dokaz pogledajte ovdje: <https://math.stackexchange.com/q/2083629/273854>.
- 7 Dokaz pogledajte ovdje: <https://math.stackexchange.com/q/3189729/273854>
- 8 Vidi [Bishop, 2006] (dio 4.3.3) ili [Murphy, 2012] (dio 8.3.1).

- [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 jednačbi 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 osmislili 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](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](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 snippetima 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 dvodimenzijalnim vektorom kao izlazom:  $(\sigma(\mathbf{w}^T \mathbf{x}), 1 - \sigma(\mathbf{w}^T \mathbf{x}))$ . 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>
- [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  $\mathbf{w}^T \phi(\mathbf{x})$  kod logističke regresije, odnosno vrijednosti  $\mathbf{w}_k^T \phi(\mathbf{x})$  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.