Logistička regresija rijetkih ulaznih podataka

Dean, Renato

Master's thesis / Diplomski rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Department of Mathematics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za matematiku**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:356254

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-23**

Repository / Repozitorij:

Repository of School of Applied Mathematics and Computer Science

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku

Renato Dean

Logistička regresija rijetkih ulaznih podataka

Diplomski rad

Osijek, 2019.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku

Renato Dean

Logistička regresija rijetkih ulaznih podataka

Diplomski rad

Mentor: doc. dr. sc. Slobodan Jelić

Osijek, 2019.

Sadržaj

1 Uvod **1**

2 Algoritmi nadziranog učenja **2**
2.1 Linearna regresija . . . 2
2.1.1 Motivacija . . . 2
2.1.2 Funkcija pogreške i konveksnost . . . 3
2.1.3 Gradijentna metoda . . . 4
2.1.4 Linearna regresija više varijabli . . . 6
2.1.5 Praktični savjeti za poboljšanje modela linearne regresije . . . 8
2.2 Logistička regresija . . . 10
2.2.1 Binarna klasifikacija . . . 10
2.2.2 Funkcija pogreške i određivanje optimalnih parametara θ . . . 11
2.2.3 Multinomna klasifikacija . . . 14

3 Primjena logističke regresije na analizu slika s magnetske rezonance **17**
3.1 Magnetska rezonanca . . . 17
3.2 Baza MRNet . . . 18
3.3 Implementacija u Pythonu . . . 19

Zaključak **23**

Literatura **25**

Prilog 1 **26**

Sažetak **32**

Abstract **32**

Životopis **33**

Poglavlje 1

Uvod

Strojno učenje nema strogu definiciju. Arthur Samuel 1959. je godine opisao strojno učenje kao područje znanosti koje računalima omogućuje učenje bez eksplicitnog programiranja. Napravio je program za igranje društvene igre dama (eng. checkers) koji je imao i mogućnost igranja računala protiv samoga sebe. Nakon više desetaka tisuća odigranih partija, analizom pozicija koje su dovodile do dobitka te pozicija koje su dovodile do gubitka, računalo je poboljšalo svoju igru te je uspješno pobijedilo i samog Samuela.

Nešto precizniju definiciju predložio je Tom Mitchell 1998. godine. Definirao je dobro postavljeni problem učenja na sljedeći način: program uči iz iskustva (eng. experience) E zadatak (eng. task) T uspjehom (eng. performance) P ako se njegov uspjeh na zadatku poboljšava iskustvom.

Zamislimo da ručno razvrstavamo mailove u kategorije željene i neželjene pošte (eng. spam) te da imamo program koji gleda kako to radimo. Tada bi zadatak T bio klasificiranje mailova, iskustvo E bi bilo promatranje našeg razvrstavanja mailova u te dvije kategorije, a uspjeh P bi bio udio dobro razvrstanih mailova. Kroz rad ćemo se upoznati s nekoliko tehnika nadziranog strojnog učenja:

Prvo ćemo kroz praktičan primjer objasniti linearnu regresiju jedne varijable, definirati funkciju pogreške te analizirati njezinu matematičku pozadinu dokazom nekih elementarnih teorema iz konveksne optimizacije. Nakon toga ćemo obraditi gradijentnu metodu za lokalnu optimizaciju i primijeniti ju na linearnu regresiju. Potom, bavit ćemo se linearnom regresijom više varijabli te ćemo sve zaokružiti s nekoliko praktičnih savjeta prilikom implementacije tih metoda u nekom od programskih jezika.

Zatim ćemo se baviti problemima klasifikacije. Upoznat ćemo se s logističkom regresijom i objasnit ćemo njezinu matematičku pozadinu te ćemo ju primijeniti na binarnu i multinomnu klasifikaciju podataka. Nakon toga ćemo dati još jedan pristup multinomnoj klasifikaciji podataka koji se zove softmax regresija te ćemo pokazati njezinu povezanost s logističkom regresijom. Konačno, na praktičnom ćemo projektu pokazati primjenu logističke regresije kao klasifikatora: detektirat ćemo potencijalna oštećenja koljena na podacima iz baze MRNet Sveučilišta Stanford koristeći Pythonovu biblioteku Scikit-learn.

Poglavlje 2

Algoritmi nadziranog učenja

2.1 Linearna regresija

2.1.1 Motivacija

Pretpostavimo da nam je dan skup od m podataka (eng. dataset) koji se sastoji od kvadrature kuća (u kvadratnim stopama) te njihovim pripadnim cijenama (u tisućama dolara):

Kvadratura	Cijena
2104	460
1416	232
1534	315
852	178
⋮	⋮

Slika 2.1: Ovisnost cijene kuće o jednom svojstvu

Trebamo definirati model funkciju koja bi na temelju poznavanja kvadrature kuće mogla predvidjeti njezinu cijenu. Navedenu funkciju ćemo označavati sa h_θ. Sa $x^{(i)}$ ćemo označavati vektor ulaznih (poznatih) parametara i-tog podatka, a sa $y^{(i)}$ pripadni rezultat. Ukoliko primijenimo ove oznake na podatke iz tablice na slici 2.1 dobivamo $x^{(1)} = 2104, y^{(1)} = 460, x^{(2)} = 1416, y^{(2)} = 232$ itd.

Uz pretpostavku da cijena kuće linearno ovisi o njezinoj kvadraturi, želimo pronaći model funkciju oblika

$$h_\theta(x) = \theta_0 + \theta_1 x,$$

sa svojstvom da je suma srednjih kvadratnih odstupanja minimalna, tj. želimo pronaći

$$\min_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2.1}$$

Funkcija koja se minimizira u izrazu (2.1) se naziva funkcija pogreške (eng. error function) i označava se sa $J(\theta_0, \theta_1)$.

2.1.2 Funkcija pogreške i konveksnost

Teorem 2.1.1. *Funkcija* $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$ *je konveksna.*

Dokaz. Izračunajmo Hessijan H = $[h_{ij}]$ funkcije $J(\theta_0, \theta_1)$:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}),$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)},$$

$$\frac{\partial}{\partial \theta_0 \partial \theta_0} J(\theta_0, \theta_1) = 1,$$

$$\frac{\partial}{\partial \theta_1 \partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0 \partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} x^{(i)},$$

$$\frac{\partial}{\partial \theta_1 \partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)}\right)^2,$$

tj.

$$H = \frac{1}{m} \begin{bmatrix} m & \sum_{i=1}^{m} x^{(i)} \\ \sum_{i=1}^{m} x^{(i)} & \sum_{i=1}^{m} \left(x^{(i)}\right)^2 \end{bmatrix}.$$

Ako je funkcija konveksna, njezin je Hessijan u svakoj točki domene pozitivno semidefinitna matrica. Prema Sylvesterovom kriteriju, za dokaz pozitivne semidefinitnosti matrice dovoljno je pokazati da su sve glavne minore nenegativne. Zaista,

$$|h_{11}| = m > 0,$$

$$|h_{22}| = \sum_{i=1}^{m} \left(x^{(i)}\right)^2 \geq 0,$$

$$|H| = m \sum_{i=1}^{m} \left(x^{(i)}\right)^2 - \left(\sum_{i=1}^{m} x^{(i)} \right)^2 \geq 0.$$

Posljednja nejednakost direktna je posljedica CSB nejednakosti. □

Za konveksne funkcije također vrijedi i sljedeći teorem:

Teorem 2.1.2. *Neka je $f : D \to \mathbb{R}$ konveksna funkcija definirana na konveksnom skupu $D \subseteq \mathbb{R}^n$. Ako je x^* lokalni minimizator funkcije f, tada je x^* ujedno i globalni minimizator funkcije f.*

Dokaz. Kako je x^* lokalni minimizator funkcije f, postoji okolina $E(x^*)$ takva da je $f(x) \geq f(x^*)$, za sve $x \in E(x^*) \cap D$. Pretpostavimo suprotno, tj. da x^* nije globalni minimizator funkcije f. To znači da postoji $z \in D$, takav da je $f(z) < f(x^*)$. Koristeći ovu nejednakost i svojstvo konveksnosti funkcije f, za proizvoljni $\alpha \in [0, 1]$ dobivamo:

$$f(\alpha z + (1-\alpha)x^*) \leq \alpha f(z) + (1-\alpha)f(x^*) < \alpha f(x^*) + (1-\alpha)f(x^*) = f(x^*).$$

Odaberimo $\alpha^+ \in [0, 1]$ u blizini nule takav da je $\alpha^+ z + (1-\alpha^+)x^* \in E(x^*) \cap D$. Prema prethodnoj nejednakosti i za taj α^+ vrijedi

$$f(\alpha^+ z + (1-\alpha^+)x^*) < f(x^*),$$

što znači da x^* nije lokalni minimizator funkcije f jer u točki $\alpha^+ z + (1-\alpha^+)x^*$ funkcija f poprima manju vrijednost, što je u kontradikciji s polaznom pretpostavkom te je teorem dokazan. □

Ovaj teorem nam je vrlo koristan jer pojednostavljujemo inače složen problem globalne optimizacije na jednostavniji problem lokalne optimizacije.

2.1.3 Gradijentna metoda

Minimizacijski problem vrlo često ne možemo riješiti egzaktno te ga u tom slučaju nastojimo riješiti nekom iterativnom numeričkom metodom. U iterativnim metodama obično polazimo od neke aproksimacije x^k za koju želimo odrediti vektor $p^k \neq 0$ te $\alpha_k > 0$ tako da vrijedi

$$f(x^k + \alpha_k p^k) < f(x^k), k = 0, 1, 2, \ldots$$

Na taj način dobivamo niz aproksimacija

$$x^{k+1} = x^k + \alpha_k p^k,$$

gdje $p^k \in \mathbb{R}^n$ zovemo vektor smjera , a $\alpha_k > 0$ duljina koraka u smjeru vektora p^k. Ako u nekoj iteraciji nismo u mogućnosti pronaći vektor smjera i duljinu koraka s kojim možemo smanjiti vrijednost funkcije f, za očekivati je da je iterativni postupak pronašao stacionarnu točku funkcije f. Ako je uz to funkcija f konveksna, točka koju smo na taj način odredili ujedno je globalni minimizator funkcije f. Sljedeća lema govori kako treba birati vektor smjera te korak kako bismo postigli sniženje vrijednosti minimizirajuće funkcije:

Teorem 2.1.3. *Neka je $f : D \to \mathbb{R}, D \subseteq \mathbb{R}^n$ diferencijabilna funkcija na konveksnom skupu D. Ako za neku točku $x \in int(D)$ i neki $p \in \mathbb{R}^n$ vrijedi $(\nabla f(x))^T p < 0$, tada postoji $\delta > 0$ takav da je*

$$f(x + \alpha p) < f(x), \forall \alpha \in (0, \delta).$$

Dokaz. Definirajmo pomoćnu funkciju $\phi : [0, +\infty) \to \mathbb{R}, \phi(\alpha) = f(x + \alpha p)$ za koju je

$$\phi'(0) = \lim_{\alpha \to 0^+} \frac{\phi(\alpha) - \phi(0)}{\alpha} = \lim_{\alpha \to 0^+} \frac{f(x + \alpha p) - f(x)}{\alpha p} p = (\nabla f(x))^T p.$$

Kako je $(\nabla f(x))^T p < 0$, postoji $\delta > 0$ takav da je $x + \alpha p \in D$ i $f(x + \alpha p) - f(x) < 0$ za sve $\alpha \in (0, \delta)$. □

Za vektor $p^k = -\nabla f(x^k)$ dobivamo gradijentnu metodu.

Ukoliko želimo minimizirati funkciju iz minimizacijskog problema (2.1) gradijentnom metodom, potrebno je izabrati početnu aproksimaciju (npr. $(0, 0)$), izračunati njezine parcijalne derivacije te odabrati korak (npr. $\alpha = 1$). Parcijalne derivacije smo već izračunali u dokazu Teorema 2.1.1:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}.$$

Sada imamo sve potrebno kako bismo sastavili pseudokod za **linearnu regresiju jedne varijable** koja koristi gradijentnu metodu kao metodu određivanja optimalnih parametara θ:

Algorithm 1 Linearna regresija jedne varijable

Require: $dataset(x^{(i)}, y^{(i)}), i = 1, \ldots, m, \quad \alpha > 0, \quad tol > 0$
  $\theta_j \leftarrow 0, \quad j = 0, 1$
  while $||\nabla J(\theta_0, \theta_1)|| \geq tol$ **do**
    $t_0 \leftarrow \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$
    $t_1 \leftarrow \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$
    $\theta_j \leftarrow t_j, \quad j = 0, 1$
  end while
  return $[\theta_0 \quad \theta_1]^T$

2.1.4 Linearna regresija više varijabli

Pretpostavimo da nam je dan sljedeći skup podataka, pri čemu je kvadratura kuće dana u kvadratnim stopama, starost kuće u godinama, a cijena kuće u tisućama dolara:

Kvadratura	Broj spavaćih soba	Broj katova	Starost kuće	Cijena kuće
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
⋮	⋮	⋮	⋮	⋮

Slika 2.2: Ovisnost cijene kuće o 4 svojstva

Uz pretpostavku da cijena kuće ovisi linearno o navedenim svojstvima, želimo pronaći model funkciju koja bi mogla predvidjeti njezinu cijenu. Uvedimo još oznaka: sa n ćemo označavati broj svojstava, a sa $x_j^{(i)}$ ćemo označavati j-to svojstvo i- tog podatka. Primjenom ovih oznaka na prethodnu tablicu dobivamo $n = 4, x^{(2)} = [1416 \quad 3 \quad 2 \quad 40]^T, x_3^{(2)} = 2.$

U ovom slučaju, model funkcija biti će

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n, \tag{2.2}$$

a ukoliko dodamo još jedno, nulto svojstvo i postavimo ga na vrijednost 1, $(x_0^{(i)} = 1, \forall i = 1, \ldots, m)$, funkciju (2.2) možemo zapisati u vektorskom obliku kao

$$h_\theta(x) = \theta^T x,$$

$$\theta = [\theta_0 \quad \theta_1 \quad \ldots \quad \theta_n]^T \in \mathbb{R}^{n+1},$$

$$x = [x_0 \quad x_1 \quad \ldots \quad x_n]^T \in \mathbb{R}^{n+1},$$

Želimo pronaći optimalne parametre vektora θ koji će biti rješenje minimizacijskog problema

$$\min_{\theta_0,\theta_1,\ldots,\theta_n} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \min_{\theta_0,\theta_1,\ldots,\theta_n} J(\theta). \tag{2.3}$$

Ukoliko izračunamo parcijalne derivacije funkcije $J(\theta)$ dobit ćemo

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)},$$

a istim pristupom kao u potpotpoglavlju 2.1.2 možemo dokazati i njezinu konveksnost. Uočimo da ukoliko je $\theta = [\theta_0 \quad \theta_1] \in \mathbb{R}^2$, dobivamo iste izraze kao u linearnoj regresiji jedne varijable.

Algorithm 2 Linearna regresija više varijabli

Require: $dataset(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, m, \quad \alpha > 0, \quad tol > 0$

$\theta_j \leftarrow 0, \quad \forall j = 0, \ldots, n$

while $||\nabla J(\theta)|| \geq tol$ **do**

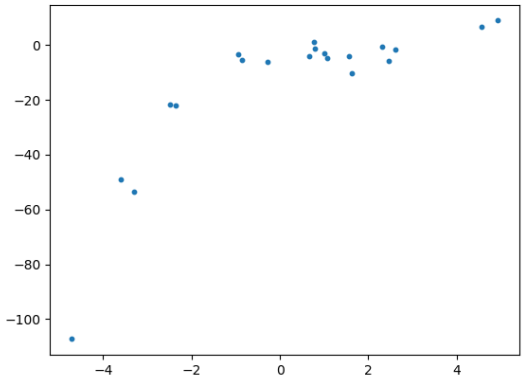
$t_j \leftarrow \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}, \quad \forall j = 0, \ldots, n$

$\theta_j \leftarrow t_j, \quad \forall j = 0, \ldots, n$

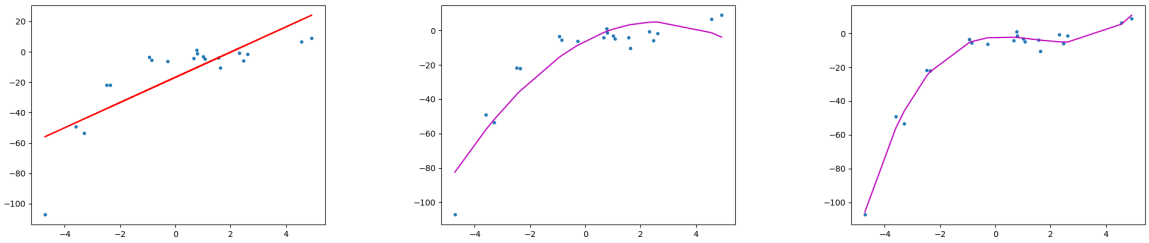
end while

return $\theta = [\theta_0 \quad \theta_1 \quad \ldots \quad \theta_n]^T$

Uz linearnu regresiju više varijabli je usko povezana i **polinomijalna regresija**. Na slici 2.3 je prikazan skup podataka, a na slici 2.4 su prikazani odgovarajući modeli polinomijalnih regresija:



Slika 2.3: Nasumično generirani podaci



Slika 2.4: Linearni, kvadratni i kubični modeli

Primijetimo da pravac sa slike 2.4 nije baš najbolje opisao ponašanje podataka, a povećanjem stupnja polinoma dobili smo poboljšanje modela. Daljnjim povećanjem stupnja polinoma možemo još bolje opisati postojeće podatke, no s time treba biti oprezan. Što je polinom višeg stupnja, to će bolje opisati postojeće podatke, ali će lošije reagirati na nove podatke, tj. dogodit će se **prenaučenost** (eng. overfitting) modela. Ukoliko želimo pronaći npr. kvadratnu funkciju $h_\theta(x)$ koja opisuje podatke, to

znači da tražimo

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2,$$

a ukoliko uvedemo supstituciju

$$t_1 = x, \quad t_2 = x^2,$$

dobivamo linearnu regresiju s dvije varijable.

2.1.5 Praktični savjeti za poboljšanje modela linearne regresije

Ukoliko imamo više ulaznih parametara (svojstava, feature-a), bilo bi ih dobro skalirati, tj. njihove vrijednosti bi trebale biti sličnog raspona. Npr, ukoliko nam je raspon za kvadraturu kuće od 0-2000 kvadratnih stopa, a broj spavaćih soba od 1-5, metoda gradijentnog spusta će dugo tražiti put do globalnog minimuma. U praksi se pokazalo dobrim imati svojstva približno u $[-1, 1]$. Uobičajen pristup modifikacije svojstava je korištenjem **normalizacije**, tj.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}, \quad \forall i = 1, \ldots, m, \quad \forall j = 1, \ldots, n.$$

Nadalje, trebali bismo biti sigurni da vrijednost funkcije $J(\theta)$ opada rastom broja iteracija, tj. trebamo nacrtati njezin graf u ovisnosti o broju iteracija. Ukoliko vrijednost $J(\theta)$ ne opada, vjerojatno smo za korak gradijentnog spusta α uzeli prevelik broj. Za dovoljno mali α, gradijentna metoda će konvergirati. Međutim, ukoliko je α jako mal, konvergencija će biti jako spora.

Jedan od pristupa je pristup pokušaja i pogreške, tj. uzmemo više različitih parametara α i pogledamo ponašanje funkcije $J(\theta)$ te odaberemo onaj za koji uz najmanje iteracija dobivamo konvergenciju. Možemo krenuti od $\alpha = 0.001$, zatim $\alpha = 0.01$, $\alpha = 0.1$ te konačno $\alpha = 1$. Nakon što smo identificirali dvije vrijednosti, jednu za koju algoritam radi, ali presporo te drugu za koju metoda ne konvergira, uzmemo manju vrijednost te ju utrostručujemo dok metoda ne prestane konvergirati i zadržimo se na posljednjoj vrijednosti za koju algoritam konvergira.

Ukoliko imamo malen broj svojstava ($n < 1000$), određivanju optimalnih parametara vektora θ možemo pristupiti i analitički: Prvo sastavimo tzv. matricu dizajna $X \in \mathbb{R}^{m\times(n+1)}$ čiji i- ti redak čini $x^{(i)}$. Ukoliko definiramo θ kao $\theta = [\theta_0 \quad \theta_1 \quad \ldots \quad \theta_n]^T \in \mathbb{R}^{n+1}$, tada funkciju $J(\theta)$ možemo zapisati matrično kao

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y).$$

Tražimo njezin minimum, tj. trebamo izjednačiti njezinu derivaciju s nulom. U tu svrhu možemo odbaciti konstantni dio $\frac{1}{2m}$. Pojednostavljivanjem dobivamo:

$$J(\theta) = ((X\theta)^T - y^T)(X\theta - y),$$

$$J(\theta) = (X\theta)^T X\theta - (X\theta)^T y - y^T(X\theta) + y^T y.$$

Jer su vektori $X\theta$ i y istih dimenzija, nije bitan redoslijed njihovog množenja te možemo zbrojiti drugi i treći član s desne strane jednakosti:

$$J(\theta) = \theta^T X^T X\theta - 2(X\theta)^T y + y^T y.$$

Ovako zapisanu funkciju $J(\theta)$ možemo derivirati:

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0,$$

tj.

$$X^T X\theta = X^T y.$$

Uz pretpostavku da je matrica $X^T X$ invertibilna, dobivamo

$$\theta = (X^T X)^{-1} X^T y.$$

Ovaj pristup se naziva **sustavom normalnih jednadžbi**. Kao što vidimo, ovdje ne moramo određivati nikakav α kao u gradijentnoj metodi niti trebamo iterirati, međutim, trebamo izračunati $(X^T X)^{-1}$, a taj izračun će raditi u kubičnom vremenu. Nadalje, može se dogoditi da je ta matrica singularna. Jedan od mogućih razloga singularnosti je taj da su neka svojstva linearno zavisna (npr. imamo podatak o kvadraturi kuće u više mjernih jedinica). Može se dogoditi i da imamo prevelik broj svojstava s obzirom na broj podataka (npr. $n = 100, m = 10$). Postoje 2 načina za rješavanje tih problema, redukcija broja svojstava te regularizacija. U redukciji broja svojstava, sami biramo koje ćemo podatke odbaciti, a u regularizaciji zadržavamo sva svojstva, ali ograničavamo parametar θ. Konkretno, modificirat ćemo funkciju cijene te će ona izgledati ovako:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right].$$

Treba izabrati $\lambda \in \mathbb{R}$ takav da pojedini parametri θ_j pomnoženi s njim teže nuli, tj. želimo dobiti polinom nižeg stupnja koji modelira podatke. Zbog promjene funkcije $J(\theta)$, moramo ažurirati i algoritam gradijentnog spusta te će on tada izgledati ovako:

Algorithm 3 Regularizirana linearna regresija više varijabli

Require: $dataset(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, m, \quad \alpha > 0, \quad tol > 0, \quad \lambda > 0$
  $\theta_j \leftarrow 0, \quad \forall j = 0, \ldots, n$
  while $||\nabla J(\theta)|| \geq tol$ **do**
    $t_j \leftarrow \theta_j(1 - \alpha\frac{\lambda}{m}) - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}, \quad \forall j = 0, \ldots, n$
    $\theta_j \leftarrow t_j, \quad \forall j = 0, \ldots, n$
  end while
  return $\theta = [\theta_0 \quad \theta_1 \quad \ldots \quad \theta_n]^T$

Slična ideja se može primijeniti i u sustavu normalnih jednadžbi: izbjeći ćemo singularnost matrice $X^T X$ tako da definiramo matricu

$$A = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)},$$

koju ćemo s regularizacijskim parametrom λ dodati singularnoj matrici $X^T X$, a novo rješenje je tada

$$\theta = (X^T X + \lambda A)^{-1} X^T y.$$

Ovakva regularizacija se naziva l_2 regularizacija. Postoji i l_1 regularizacija pod nazivom LASSO (least absolute shrinkage and selection operator) te elastic-net koji je kombinacija l_1 i l_2 regularizacije.

2.2 Logistička regresija

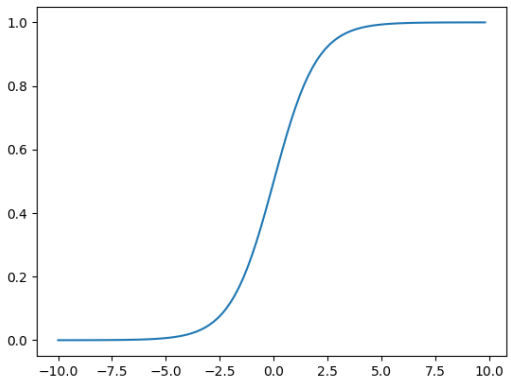
2.2.1 Binarna klasifikacija

Neka nam je dan skup mailova za koje želimo reći jesu li spam ili ne, tj. želimo da je $y^{(i)} \in \{0, 1\}, \forall i = 1, \ldots, m$. U tu svrhu, potrebno je definirati funkciju koja svaki element domene preslika u neki broj iz $[0, 1]$. Jedna od funkcija koja to radi je **logistička** funkcija. Ona je definirana kao

$$\sigma : \mathbb{R} \to (0, 1)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

a njezin graf prikazan je na slici 2.5:



Slika 2.5: Graf logističke funkcije

Vrijednosti funkcije (2.2) su mogle ići daleko izvan $[0, 1]$, no ukoliko ju komponiramo s logističkom funkcijom, tj. napravimo $\sigma(h_\theta(x))$ dobivamo

$$\sigma(h_\theta(x)) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}.$$

Zbog definicije logističke funkcije, sigurni smo da su vrijednosti ovako dobivene funkcije u $[0, 1]$. U ovom dijelu rada ćemo stoga koristiti oznaku $h_\theta(x)$ upravo za navedenu kompoziciju, tj.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}. \tag{2.4}$$

Vrijednost u pojedinoj točki x iz njezine domene interpretirat ćemo kao vjerojatnost da je $y = 1$ uz uvjet x parametrizirano sa θ, što ćemo zapisivati na sljedeći način:

$$P(y = 1|x; \theta).$$

Primjenom vjerojatnosti suprotnog događaja možemo izraziti i

$$P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta).$$

Iz grafa prikazanog na slici 2.5 možemo uočiti i da je

$$h_\theta(x) \geq 0.5 \Leftrightarrow x \geq 0,$$

stoga nam je za klasifikaciju elementa x dovoljno izračunati predznak od $\theta^T x$. Krivulja koja razdvaja područja za koje je $\theta^T x \geq 0$ te $\theta^T x < 0$ naziva se granica odlučivanja (eng. decision boundary).

Uvodno razmatranje zaključit ćemo primjerom. Pretpostavimo da imamo $n = 2$ svojstva, da je

$$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

te da smo odredili vektor optimalnih parametara

$$\theta = [-3 \quad 1 \quad 1]^T.$$

Ukoliko uvrstimo θ dobivamo $-3 + x_1 + x_2$. Dakle, ukoliko je $-3 + x_1 + x_2 \geq 0$, reći ćemo da je $y = 1$, dok ćemo u suprotnom reći da je $y = 0$. U ovom slučaju granica odlučivanja je pravac, tj x_i su linearni, a uvrštavanjem nelinearnih članova x_i, za granicu odlučivanja možemo dobiti i neke složenije krivulje. Ukoliko umjesto x_1 i x_2 stavimo njihove kvadrate, za granicu odlučivanja dobivamo $-3 + x_1^2 + x_2^2 \geq 0$, tj. kružnicu.

2.2.2 Funkcija pogreške i određivanje optimalnih parametara θ

Za funkciju pogreške nećemo moći koristiti funkciju $J(\theta)$ iz minimizacijskog problema (2.3). Kako bismo to obrazložili, prvo dokažimo jedno svojstvo konveksnih derivabilnih funkcija.

Teorem 2.2.1 (Gradijentna nejednakost)**.** *Ako je $f : \mathbb{R}^n \to \mathbb{R}$ konveksna diferencijabilna funkcija, onda za nju vrijedi*

$$f(u) \geq f(v) + \nabla f(v)^T (u - v), \forall u, v \in \mathbb{R}^n.$$

Dokaz. Po definiciji konveksne funkcije,

$$\forall u, v \in \mathbb{R}^n, \forall \lambda \in [0, 1] \Rightarrow f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda) f(v).$$

Zapišimo tu nejednakost na drugi način za $\lambda \in (0, 1)$:

$$\lambda f(u) \geq f(v + \lambda(u - v)) - (1 - \lambda) f(v),$$

što nakon dijeljenja s $\lambda > 0$ možemo zapisati kao:

$$f(u) \geq f(v) + \frac{f(v + \lambda(u - v)) - f(v)}{\lambda(u - v)}(u - v).$$

Ukoliko sada pustimo $\lambda \to 0^+$, dobivamo tvrdnju teorema. □

Teorem 2.2.2. *Funkcija* $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$*, pri čemu je* h_θ *funkcija (2.4), nije konveksna.*

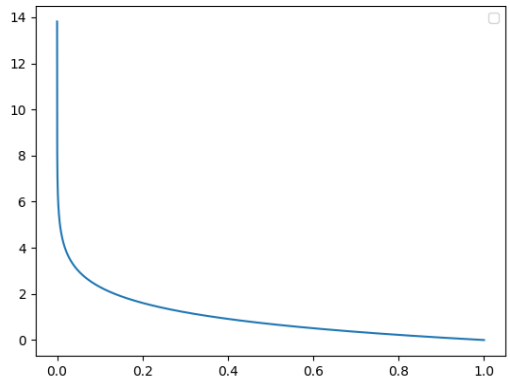
Dokaz. Koristimo gradijentnu nejednakost. Dokaz ćemo napraviti u skupu $\mathbb{R}$. Derivirajmo funkciju $h_\theta(x)$:

$$(h_\theta(x))' = h_\theta(x)(1 - h_\theta(x)).$$

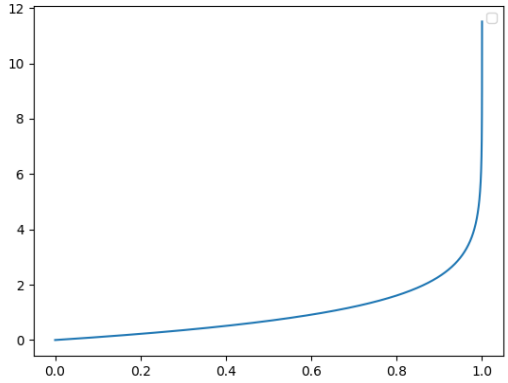
Za $u = 0, v = 1 \Rightarrow h_\theta(u) = 0.5, (h_\theta(u))' = 0.25$. Ukoliko sada uvrstimo te brojeve u gradijentnu nejednakost dobivamo kontradikciju, tj. h_θ nije konveksna, što znači da ni $J(\theta)$ nije konveksna te je time teorem dokazan. □

Očito nam treba neki drugi izbor za funkciju pogreške. Definirajmo sljedeću funkciju te prikažimo njezin graf:

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & y = 1 \\ -\log(1 - h_\theta(x)), & y = 0 \end{cases}.$$



Slika 2.6: Graf Cost funkcije za $y = 1$



Slika 2.7: Graf Cost funkcije za $y = 0$

Cilj funkcije je strogo penaliziranje krivih klasifikacija. Ukoliko Cost funkciju zapišemo na drugi način,

$$\text{Cost}(h_\theta(x), y) = -y\log(h_\theta(x)) - (1 - y)\log(1 - h_\theta(x)),$$

možemo definirati funkciju $J(\theta)$ kao

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{Cost}(h_\theta(x^{(i)}), y^{(i)}),$$

odnosno

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right] \tag{2.5}$$

Kako bismo mogli primijeniti neku od lokalnih optimizacijskih metoda, kao što je npr. gradijentna metoda, trebamo dokazati konveksnost funkcije $J(\theta)$. O tome govori sljedeći teorem:

Teorem 2.2.3. *Funkcija* $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$ *je konveksna.*

Dokaz. Konstantni dio $\frac{1}{m}$ ne utječe na konveksnost te ćemo ga stoga zanemariti i minimizirat ćemo

$$J(\theta) = \sum_{i=1}^{m}(y^{(i)}[-\log(h_\theta(x^{(i)}))] + (1-y^{(i)})[-\log(1-h_\theta(x^{(i)}))]).$$

Ukoliko pokažemo da su $-\log(h_\theta(x^{(i)}))$ i $-\log(1-h_\theta(x^{(i)}))$ konveksne, tada će i njihova pozitivna linearna kombinacija biti konveksna. Pokažimo prvo da je $-\log(h_\theta(x)) = \log(1+e^{-\theta^T x})$ konveksna, izračunajmo njezin Hessijan:

$$\nabla_\theta[\log(1+e^{-\theta^T x})] = \left(\frac{-e^{-\theta^T x}}{1+e^{-\theta^T x}}\right)x = \left(\frac{1}{1+e^{-\theta^T x}} - 1\right)x = (h_\theta(x)-1)x,$$

$$\nabla^2_\theta[\log(1+e^{-\theta^T x})] = h_\theta(x)(1-h_\theta(x))xx^T.$$

Koristeći definiciju pozitivne semidefinitnosti, treba pokazati da je

$$\forall z, \quad z^T\nabla^2_\theta[\log(1+e^{-\theta^T x})]z \geq 0.$$

I zaista,

$$z^T\nabla^2_\theta[\log(1+e^{-\theta^T x})]z = z^T[h_\theta(x)(1-h_\theta(x))xx^T]z = h_\theta(x)(1-h_\theta(x))(x^Tz)^2 \geq 0.$$

Pokažimo sada da je $-\log(1-h_\theta(x^{(i)})) = \theta^T x + \log(1+e^{-\theta^T x})$ konveksna.

$$\nabla_\theta[\theta^T x + \log(1+e^{-\theta^T x})] = x + \nabla_\theta[\log(1+e^{-\theta^T x})],$$

$$\nabla^2_\theta[\theta^T x + \log(1+e^{-\theta^T x})] = \nabla_\theta[x + \nabla_\theta[\log(1+e^{-\theta^T x})]] = \nabla^2_\theta[\log(1+e^{-\theta^T x})],$$

a za taj Hessijan smo već dokazali pozitivnu definitnost. □

Zbog konveksnosti $J(\theta)$, možemo tražiti njezin minimum primjenom gradijentne metode. Ukoliko izračunamo $\frac{\partial}{\partial\theta_j}J(\theta)$, dobit ćemo isti izraz kao i u linearnoj regresiji više varijabli (do na konstantu):

$$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)},$$

stoga će algoritam za logističku regresiju izgledati ovako:

Algorithm 4 Logistička regresija

Require: $dataset(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, m, \quad \alpha > 0, \quad tol > 0$
  $\theta_j \leftarrow 0, \quad \forall j = 0, \ldots, n$
  while $||\nabla J(\theta)|| \geq tol$ **do**
    $t_j \leftarrow \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}, \quad \forall j = 0, \ldots, n$
    $\theta_j \leftarrow t_j, \quad \forall j = 0, \ldots, n$
  end while
  return $\theta = [\theta_0 \quad \theta_1 \quad \ldots \quad \theta_n]^T$

Treba biti oprezan kako se ne bi pomiješale definicije od h_θ. U ovom slučaju je to logistička, a ne linearna funkcija.

2.2.3 Multinomna klasifikacija

Pretpostavit ćemo da nam je dan skup podataka $(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, m$ te da je $y^{(i)} \in \{1, 2, \ldots, K\}$. Jedan od pristupa multinomne klasifikacije objasnit ćemo na sljedećem primjeru: neka nam se podaci sastoje od slika rukom pisanih znamenaka $1, 2$ i 3. Logističkom regresijom ćemo redom napraviti modele koji će detektirati pojedinu znamenku. Slike s tom znamenkom proglasit ćemo pozitivnom klasom, a sve ostale podatke proglasit ćemo negativnom klasom.

U ovom primjeru, morat ćemo napraviti 3 modela: prvi će reći da je znamenka 1 u pozitivnoj klasi, a znamenke 2 i 3 u negativnoj. Označimo ga sa $h_\theta^{(1)}$.

Drugi model proglasit će znamenku 2 elementom pozitivne klase, a znamenke 1 i 3 elementima negativne klase. Označimo ga sa $h_\theta^{(2)}$.

Treći model proglasit će znamenku 3 elementom pozitivne klase, a znamenke 1 i 2 elementima negativne klase. Označimo ga sa $h_\theta^{(3)}$.

Nakon toga, za svaki podatak $x^{(i)}$ evaluiramo vrijednost sve 3 funkcije i izaberemo onaj j za koji je $h_\theta^{(j)}$ maksimalan.

Taj pristup često može biti spor. Ideja bržeg pristupa je sljedeća: za zadani x želimo da model funkcija procijeni vjerojatnost

$$P(y = k|x), \quad \forall k = 1, \ldots K.$$

Za rezultat procjene želimo da bude K-dimenzionalni vektor čija je suma elemenata jednaka 1. Konkretno, hipoteza će biti

$$h_\theta(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ P(y = 2|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} e^{\theta^{(j)T} x}} \begin{bmatrix} e^{\theta^{(1)T} x} \\ e^{\theta^{(2)T} x} \\ \vdots \\ e^{\theta^{(K)T} x} \end{bmatrix}.$$

Izraz $\frac{1}{\sum_{j=1}^{K} e^{\theta^{(j)T} x}}$ služi za normalizaciju, tj. osigurava da je suma elemenata navedenog vektora jednaka 1.

Vektore $\theta^{(1)T}x, \theta^{(2)T}x, \ldots, \theta^{(K)T}x \in \mathbb{R}^n$ nazivamo parametrima modela, a sa θ ćemo označavati matricu čiji su stupci navedeni vektori, tj.

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \cdots & \theta^{(K)} \\ | & | & | & | \end{bmatrix} \in \mathbb{R}^{n \times K}.$$

Kako bismo definirali funkciju pogreške, trebamo uvesti oznaku tzv. indikator funkcije:

$$1\{x\} = \begin{cases} 1, & x = \text{istina} \\ 0, & x = \text{neistina} \end{cases}.$$

Dakle, $1\{1+1=2\} = 1$, a $1\{2+2=5\} = 0$. Funkciju $J(\theta)$, koja se naziva **softmax** funkcija sada možemo definirati kao:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{e^{\theta^{(k)T}x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T}x^{(i)}}}\right]$$

Uočimo da je $J(\theta)$ generalizacija funkcije (2.3) koju možemo zapisati kao

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\} \log P(y^{(i)} = k|x^{(i)};\theta)\right].$$

U slučaju softmax funkcije je

$$P(y^{(i)} = k|x^{(i)};\theta) = \frac{e^{\theta^{(k)T}x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T}x^{(i)}}}.$$

Njezin minimum ne može se pronaći analitički, stoga joj trebamo pristupiti nekom iterativnom numeričkom metodom. Ukoliko joj izračunamo gradijent dobit ćemo

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)} = k\} - P(y^{(i)} = k|x^{(i)};\theta)\right)\right].$$

$\nabla_{\theta^{(k)}} J(\theta)$ je vektor, a njegov l- ti element je $\frac{\partial J(\theta)}{\partial \theta_{lk}}$. Sada možemo navesti pseudokod za softmax regresiju:

Algorithm 5 Softmax regresija

Require: $dataset(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, m, \quad \alpha > 0, \quad tol > 0$
  $\theta^{(k)} \leftarrow 0, \quad \forall k = 1, \ldots, K$
  while $||\nabla_{\theta^{(k)}} J(\theta)|| \geq tol, \forall k = 1, \ldots, K$ **do**
    $t^{(k)} \leftarrow \theta^{(k)} - \alpha \sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right], \quad \forall k = 1, \ldots, K$
    $\theta^{(k)} \leftarrow t^{(k)}, \quad \forall k = 1, \ldots, K$
  end while
  return $\theta = \left[\theta^{(1)T} \quad \theta^{(2)T} \quad \ldots \quad \theta^{(K)T} \right]$

Ukoliko za svaki vektor parametara $\theta^{(j)}$ oduzmemo neki fiksan vektor ψ dobit ćemo

$$P(y^{(i)} = k | x^{(i)}; \theta) = \frac{e^{(\theta^{(k)} - \psi)^T x^{(i)}}}{\sum_{j=1}^{K} e^{(\theta^{(j)} - \psi)^T x^{(i)}}} = \frac{e^{\theta^{(k)T} x^{(i)}} e^{-\psi^T x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T} x^{(i)}} e^{-\psi^T x^{(i)}}} = \frac{e^{\theta^{(k)T} x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T} x^{(i)}}},$$

što znači da postoji više parametara θ koji će nam dati isti rezultat klasifikacije, tj. model je "preparametriziran". Dodatno, ukoliko θ minimizira funkciju $J(\theta)$, tada ju minimizira i $\theta - \psi$, tj. minimizator nije jedinstven. Međutim, funkcija $J(\theta)$ je konveksna te se minimum može pronaći gradijentnom metodom. Za vektor ψ možemo uzeti bilo koji od vektora, npr. $\psi = \theta^{(k)}$, a time $\theta^{(k)}$ postaje nulvektor. Na taj način smo umanjili broj parametara koji treba optimizirati za n. Za kraj, uvjerimo se da je za $K = 2$, softmax regresija zapravo logistička regresija. Uvrštavanjem dobivamo:

$$h_\theta(x) = \frac{1}{e^{\theta^{(1)T} x} + e^{\theta^{(2)T} x}} \begin{bmatrix} e^{\theta^{(1)T} x} \\ e^{\theta^{(2)T} x} \end{bmatrix},$$

a ukoliko uvrstimo $\psi = \theta^{(2)}$:

$$h_\theta(x) = \frac{1}{e^{(\theta^{(1)} - \theta^{(2)})^T x} + e^{\vec{0}^T x}} \begin{bmatrix} e^{(\theta^{(1)} - \theta^{(2)})^T x} \\ e^{\vec{0}^T x} \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{1 + e^{(\theta^{(1)} - \theta^{(2)})^T x}} \\ \frac{1}{1 + e^{(\theta^{(1)} - \theta^{(2)})^T x}} \end{bmatrix},$$

te konačno supstitucijom $\theta := \theta^{(2)} - \theta^{(1)}$:

$$h_\theta(x) = \begin{bmatrix} 1 - \frac{1}{1 + e^{-\theta^T x}} \\ \frac{1}{1 + e^{-\theta^T x}} \end{bmatrix},$$

što odgovara vjerojatnostima iz logističke regresije.

Poglavlje 3

Primjena logističke regresije na analizu slika s magnetske rezonance

3.1 Magnetska rezonanca

Magnetska rezonanca je medicinski postupak koji koristi veliki magnet, radiovalove i računalo za stvaranje detaljne slike poprečnog presjeka unutarnjih organa. Sam skener obično nalikuje velikoj cijevi sa stolom u sredini. Pretraga se razlikuje od CT-a i X-zraka jer ne koristi potencijalno štetna ionizirajuća zračenja.

Razvoj magnetske rezonance predstavlja ogromnu prekretnicu za medicinski svijet jer liječnici, znanstvenici i istraživači sada mogu detaljno pregledati unutrašnjost ljudskog tijela koristeći neinvazivni alat. Magnetsku rezonancu možemo koristiti za detekciju

- anomalije mozga i leđne moždine
- tumora, cista i drugih anomalija u raznim dijelovima tijela
- ozljeda ili abnormalnosti zglobova, poput leđa i koljena
- određene vrste srčanih problema
- bolesti jetre i drugih trbušnih organa
- anomalije maternice kod žena
- ...

Najvažniji dijelovi opreme skenera su dva snažna magneta. Ljudsko tijelo se u velikoj mjeri sastoji od molekula vode, koje se sastoje od atoma vodika i kisika. U središtu svakog atoma nalazi se još manja čestica koja se naziva proton koja služi kao magnet i osjetljiva je na bilo koje magnetsko polje.

Molekule vode u tijelu obično su nasumično raspoređene, ali prilikom ulaska u skener, prvi magnet uzrokuje da se molekule vode poravnaju u jednom smjeru, bilo prema sjeveru ili prema jugu. Zatim se uključuje i isključuje i drugo magnetsko polje nizom brzih impulsa što uzrokuje da svaki

atom vodika promijeni svoje poravnanje kad je ono uključeno, a isključenjem se vrati u prvobitno stanje. Provodeći struju kroz zavojnice, stvara se magnetsko polje, izazivajući zvuk unutar skenera.

Iako pacijent ne može osjetiti ove promjene, uređaj ih može otkriti i, u suradnji s računalom, može stvoriti detaljnu sliku presjeka za radiologa.

Slika 3.1: Uređaj za magnetsku rezonancu

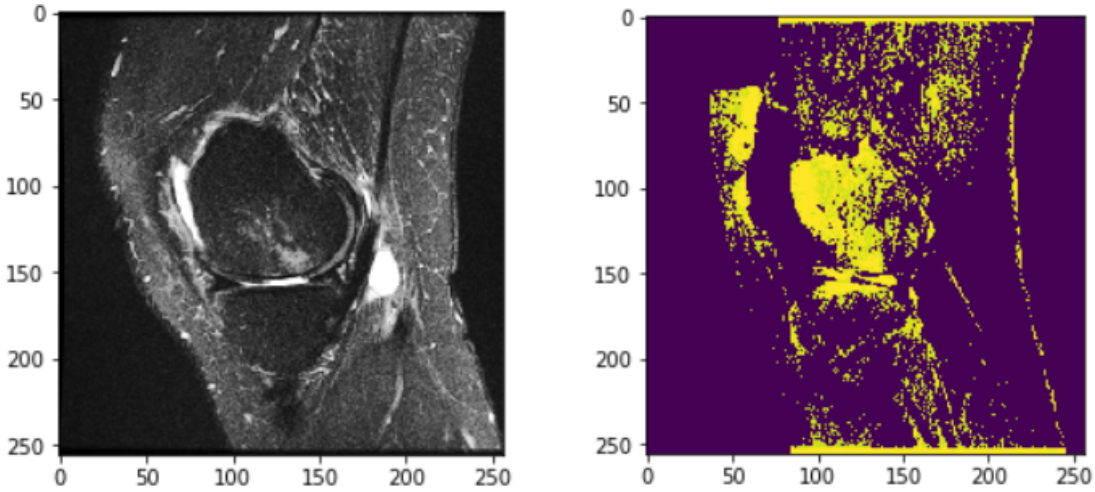
3.2 Baza MRNet

MRNet baza podataka sadrži slike koljena 1370 pacijenata s magnetske rezonance obavljene u Medicinskom centru Sveučilišta Stanford u razdoblju između 1. siječnja 2001. i 31. prosinca 2012. Kod 1104 pacijenta je detektirana abnormalnost, kod 319 pacijenata je dijagnosticirana ozljeda ACL-a (anteriornog križnog ligamenta), a kod njih 508 je detektirana ozljeda meniskusa.

Baza je podijeljena na skup podataka za treniranje, u kojem se nalaze podaci o 1130 pacijenata, skup za validiranje koji se sastoji od podataka o 120 pacijenata, te na skriveni skup koji se sastoji od podataka preostalih 120 pacijenata. MRNet baza podataka izvorno je korištena za razvoj MRNet-a, modela dubokog učenja koji može „brzo generirati precizne kliničke patološke klasifikacije pregleda koljena magnetskom rezonancom." Baza podataka je otvorenog koda (eng. open source) te postoji otvoren natječaj za modele koji mogu automatski interpretirati slike magnetske rezonance koljena.

Istraživači mogu preuzeti bazu podataka i sudjelovati u natjecanju, a predani algoritmi testirat će se na skrivenom skupu.

Podaci iz originalne baze su različitih dimenzija te je na njima izvršena redukcija volumena korištenjem nenegativne matrične faktorizacije. Kao rezultat navedenog postupka, svaka slika u bazi je dimenzija $256 \cdot 256$ piksela, te je svakom pacijentu pridruženo 15 slika. Ukupan udio ne-nul piksela je približno jednak 0.2. Na slici 3.2 je prikazan primjer originalne i transformirane slike iz baze:



Slika 3.2: Primjer originalne i segmentirane slike koljena

3.3 Implementacija u Pythonu

Za implementaciju klasifikatora koji prognozira oštećenja koljena u Pythonu možemo koristiti Scikit-learn, Pythonovu biblioteku koja sadrži alate za strojno učenje. Izgrađena je na bibliotekama NumPy, SciPy te matplotlib, a neke od njihovih primjena su brzo rješavanje optimizacijskih problema, efikasna manipulacija matrica te crtanje. Nama će u projektu trebati:

- sklearn.linear_model.LogisticRegression, koji će provesti logističku regresiju na danim podacima u matrici dizajna X te njihovim pripadajućim oznakama koji se nalaze u vektoru y,
- sklearn.preprocessing, koji će služiti skaliranju podataka u matrici dizajna X,
- sklearn.metrics, čija je svrha analiza rezultata dobivenog modela,
- sklearn.model_selection.cross_validate, čija je svrha validacija modela.

Od ostalih važnijih alata treba spomenuti i one potrebne za formiranje rijetkih (eng. sparse) matrica, a to su csr_matrix, hstack i vstack iz biblioteke scipy.sparse te alat za crtanje, matplotlib.pyplot.

Uočimo da su slike formata $15 \times 256 \times 256$, a u matrici dizajna X svaki pacijent mora biti točno jedan redak, tj. jednodimenzionalni vektor. Koristeći funkciju resize() iz biblioteke cv2, možemo slikama smanjiti volumen na $15 \times 64 \times 64$ radi uštede memorije te ih pretvoriti u jednodimenzionalni vektor koji će sadržavati $15 \cdot 64 \cdot 64 = 61440$ elemenata. Dobiveni vektor pretvaramo u sparse format koristeći scipy.sparse.csr_matrix() te ga dodajemo kao posljednji redak matrice dizajna X koristeći scipy.sparse.vstack(). Paralelno spremamo i dijagnozu odgovarajućeg pacijenta u vektor oznaka y. Ilustrirajmo kako program sprema sliku dimenzija $2 \times 2 \times 2$:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{matrix} (0,0) & 1 \\ (0,7) & 1 \end{matrix}$$

Iduće što treba napraviti je skalirati podatke iz matrice dizajna X kako bi optimizacijska metoda brže pronašla minimum. Treba biti oprezan pri odabiru metode skaliranja kako ne bismo poremetili rijetkost matrice X. Metoda koja čuva rijetkost matrice je MaxAbsScaler() iz sklearn,preprocessing, a ona će svaki element u matrici X podijeliti sa po apsolutnoj vrijednosti najvećim elementom te matrice. Npr, za matricu

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

navedenim skaliranjem dobivamo

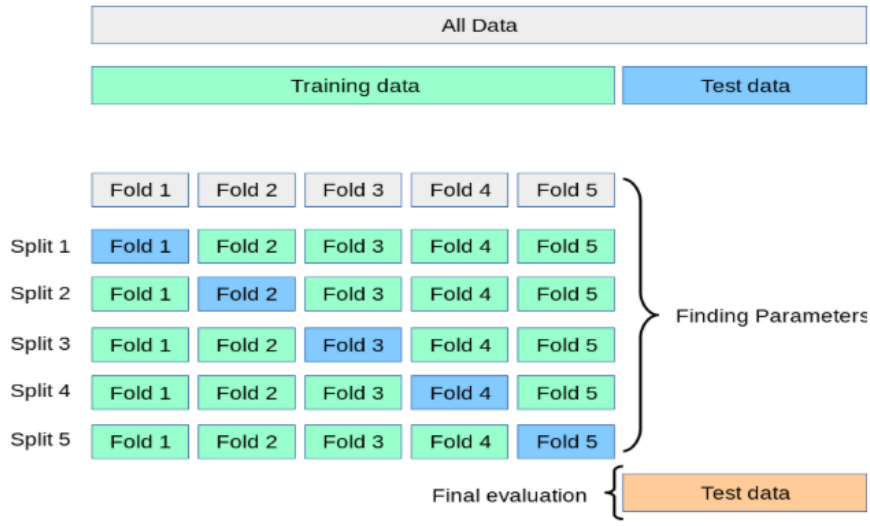
$$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Nadalje, trebamo trenirati i testirati model. Prilikom treniranja modela treba odabrati i vrstu regularizacije, veličinu regularizacijskog faktora te optimizacijsku metodu. Za rijetke matrice je najbolje odabrati $L1$ regularizaciju, a za velike skupove podataka se kao dobra optimizacijska metoda pokazala modificirana gradijentna metoda zvana SAGA (vidi [4]). Za regularizacijski faktor ćemo ostaviti pretpostavljenu vrijednost, koja je 1.

Jedan od standardnih pristupa treniranja i testiranja modela je taj da se od skupa svih podataka nasumično uzme $\frac{4}{5}$ podataka na kojima se trenira, a na ostatku se testira. Ponekad je taj pristup problematičan, jer rezultati mogu jako ovisiti o tome koji podaci su odabrani za trening, a koji za test. Bolji pristup je tzv. k-terostruka unakrsna validacija (eng. k-**fold cross validation**). Odaberemo neki prirodan broj k i podijelimo cijeli skup podataka na k dijelova (često je dobar odabir $k = 10$). Zatim sastavljamo k modela. U svakom ćemo od tih modela uzeti $k - 1$ dijelova za trening, a 1 za test. Ideja je da iskoristimo sve podatke i za trening i za test i na taj način dobijemo realniju sliku uspješnosti modela.

Na primjeru za $k = 10$, prvi model dobivamo tako da iskoristimo prvi dio podataka za test, a dijelove od 2 do 10 za trening. Drugi ćemo model dobiti tako da iskoristimo drugi dio podataka za test, a dio 1 te dijelove od 3 do 10 za trening. Analogno dobivamo i ostale modele.

Moguće je i kombiniranje navedenih pristupa. Prvo podijelimo cijeli skup podataka u nekom omjeru (npr. $4 : 1$) te na većem dijelu napravimo trening i unakrsnu validaciju i pogledamo rezultate. Nakon toga, model testiramo na ostatku podataka te usporedimo rezultate. Na slici 3.3 je grafički prikazan navedeni postupak:



Slika 3.3: k-terostruka unakrsna validacija

Prije nego što prijeđemo na pokazatelje uspješnosti modela, pogledajmo sve moguće ishode binarne klasifikacije podataka. Znamo da je prava vrijednost svakog podatka 0 ili 1 te da za svaki podatak klasifikator predviđa kojem skupu on pripada. Ishode možemo prikazati tablično, pri čemu je gornje zaglavlje predviđena vrijednost, a lijevo zaglavlje stvarna vrijednost:

	0	1
0	true negatives (TN)	false positives (FP)
1	false negatives (FN)	true positives (TP)

Matrica sastavljena od pripadnih vrijednosti te tablice se naziva matrica zbunjenosti (eng. confusion matrix).

Kada testiramo model, za svaki podatak utvrdimo točnost klasifikacije te povećamo kardinalitet odgovarajućeg skupa u tablici za 1. Jedan od osnovnih pokazatelja uspješnosti modela je točnost (eng. **accuracy**). Definira se kao

$$\text{accuracy} = \frac{|\text{TN} + \text{TP}|}{|\text{TN} + \text{FP} + \text{FN} + \text{TP}|}.$$

Ova vrijednost često može zavarati. Zamislimo da imamo skup podataka od 100 pacijenata u kojem se nalaze informacije o tome ima li pacijent rak ili ne te pretpostavimo da ih 90 nema rak i da ih 10 ima. Sastavimo trivijalni klasifikator koji će za svakog pacijenta reći da nema rak. Tada gornja tablica poprima sljedeće vrijednosti:

	0	1
0	90	0
1	10	0

Jasno je da ovakav klasifikator nema smisla, ali točnost mu je 0.9. Zbog toga se u praksi uz točnost uvode još neki pokazatelji kvalitete modela: preciznost (eng. **precision**), odaziv (eng. **recall**), **F1**, a definiraju se kao:

$$\text{precision} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FP}|},$$

$$\text{recall} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}|},$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Svi pokazatelji se nalaze u $[0, 1]$ i što su oni bliži 1, to je model bolji. Na prethodnom primjeru možemo uočiti da je točnost 0.9 i odaziv 0. Preciznost i vrijednost F1 u ovom primjeru nisu definirane, ali ih je smisleno postaviti na nulu.

Konačno, korisno je uzeti nekolicinu po apsolutnoj vrijednosti najvećih parametara θ iz dobivenog modela te ih označiti na slikama kako bismo vidjeli koja područja slike klasifikator smatra najznačanijima. Na slici 3.4 su prikazani rezultati deseterostruke unakrsne validacije na segmentiranom MRNet datasetu:

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Mean
Accuracy	0.64	0.66	0.64	0.74	0.69	0.66	0.65	0.68	0.6	0.67	0.66
Precision	0.5	0.53	0.5	0.76	0.59	0.55	0.52	0.59	0.43	0.54	0.55
Recall	0.4	0.4	0.4	0.42	0.42	0.4	0.36	0.38	0.33	0.43	0.39
F1 score	0.44	0.46	0.44	0.54	0.49	0.46	0.42	0.46	0.38	0.48	0.46

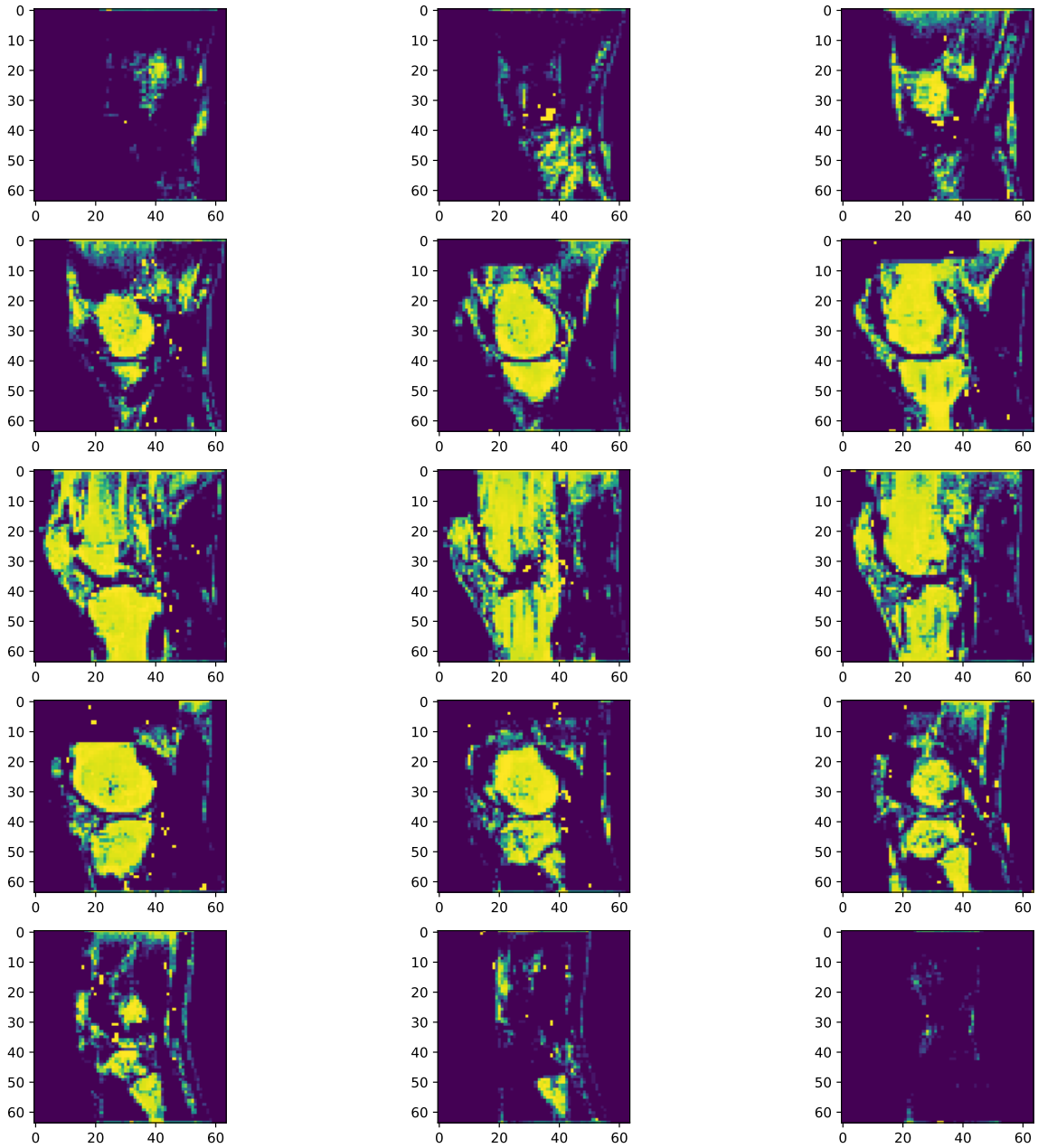
Slika 3.4: Rezultati deseterostruke unakrsne validacije

Napomena: rezultati navedeni u ovom dijelu rada se ponešto razlikuju od onih iz Priloga 1. Možemo primijetiti da klasifikator radi dosta pogrešaka jer su vrijednosti pokazatelja relativno niske. Ukoliko za skup za treniranje uzmemo cijeli skup podataka, dobivamo matricu zbunjenosti

$$\begin{bmatrix} 801 & 0 \\ 0 & 449 \end{bmatrix},$$

iz koje slijedi da su accuracy, precision i recall te F1 jednaki 1. Dakle, model je prenaučen, tj. dogodio se overfitting koji se nije uspio riješiti regularizacijom. Preostaje vidjeti koje piksele klasifikator smatra bitnima:

Slika 3.5: Najznačajniji pikseli modela



Zaključak

Rješavanje problema traženja uzoraka na slici i njihove klasifikacije je složen problem koji se u praksi uglavnom rješava primjenom dubokog učenja, tj. konvolucijskim mrežama. Ukoliko želimo pokušati primijeniti logističku regresiju kao klasifikator slika, moramo osigurati jednakost dimenzija slika te bi se pozicije bitnih svojstava na različitim slikama trebale nalaziti na približno istim lokacijama. Logistička regresija je vrlo osjetljiva na "nedotjerane" ulazne podatke. Nažalost, u praksi su podaci gotovo uvijek takvi. Model dobiven u ovom projektu bi se potencijalno mogao poboljšati dodatnim preprocesuiranjem podataka.

Literatura

[1] A. Agarwal, *Polynomial Regression*. Oct 8, 2018, https://towardsdatascience.com/polynomial-regression-bbe8b9d97491

[2] E. Bendersky, *Derivation of the Normal Equation for Linear Regression*. Dec 22, 2014, https://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression/

[3] D. R. Chittajallu, *Why is the error function minimized in logistic regression convex?* Oct 21, 2011, http://mathgotchas.blogspot.com/2011/10/why-is-error-function-minimized-in.html

[4] A. Defazio, F. Bach, S. Lacoste-Julien, *SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives*. 16 Dec 2014, https://arxiv.org/pdf/1407.0202v3.pdf

[5] P. Lam, *What to know about MRI scans*. July 24, 2018, https://www.medicalnewstoday.com/articles/146309.php

[6] A. Ng, *Machine Learning*. https://www.coursera.org/learn/machine-learning

[7] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.

[8] R. Scitovski, K. Sabo, D. Grahovac, *Globalna optimizacija*. Sveučilište Josipa Jurja Strossmayera u Osijeku – Odjel za matematiku, Osijek, 2017.

[9] *http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/*

[10] https://medium.com/syncedreview/stanford-ml-releases-mrnet-knee-mri-dataset-9f44d7621131

Prilog 1

[1]:
```
import numpy as np
from scipy.sparse import csr_matrix, vstack, hstack
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
#import seaborn as sns
from sklearn import metrics, preprocessing
import pickle
import os
import cv2
import warnings
from sklearn.model_selection import cross_val_score, cross_validate, GridSearchCV
warnings.filterwarnings("ignore")
```

[2]:
```
def loadData():
    # directory where the volumetric data is located
    volumetric_data_dir = 'input15'
    # path to metadata csv file
    metadata_csv_path = 'diagnosis'

    # names=True loads the interprets the first row of csv file as column names
    # 'i4' = 4 byte signed integer, 'U20' = unicode max 20 char string
    metadata = np.genfromtxt(metadata_csv_path, delimiter=',', names=True,
        dtype='U4, uint8')
    i = 0
    X = np.array([], dtype = np.uint8)
    y = np.array([], dtype = np.uint8)
    imgs = np.array([], dtype = np.uint8)
    print('Loading data...')
    bias = csr_matrix(np.ones(1250, dtype = np.uint8))
    for exam in metadata:
        i += 1
        vol_data_file = str(exam['ID']) + '.pck'

        vol_data_path = os.path.join(volumetric_data_dir, vol_data_file)
        # Load data from file
        with open(vol_data_path, 'rb') as file_handler: # Must use 'rb' as the␣
↪data is binary
            volumetric_data = pickle.load(file_handler)
            if volumetric_data.shape == (15,256,256):
                elems = np.array([])
                for img in volumetric_data:
                    width = 64
                    height = 64
                    dim = (width,height)
                    resized = cv2.resize(img, dim, interpolation = cv2.
↪INTER_AREA)
                    elems = np.append(elems, resized)
```

```
                    imgs = np.append(imgs, resized)
                X = vstack([X, elems]) if X.size else csr_matrix(elems)
                y = np.append(y, [exam['Dijagnoza']])
        if i % 100 == 0:
            print('Loaded {} images'.format(i))
    X = hstack([np.transpose(bias), X])
    print('{} patients loaded'.format(i))
    return X,y,imgs
```

[3]:
```
def train(x_train, y_train):
    print('Training...')
    clf = LogisticRegression(penalty = 'l1', solver = 'saga', max_iter = 20,␣
 ↪n_jobs = -1, verbose = 1).fit(x_train,y_train)
    print('Done with training')
    return clf
```

[4]:
```
def predict(model, x_test):
    print("Predicting...")
    predictions = model.predict(x_test)
    return predictions
```

[5]:
```
def validate(model,X,y):
    scoring = {'accuracy' : metrics.make_scorer(metrics.accuracy_score),
               'precision' : metrics.make_scorer(metrics.precision_score),
               'recall' : metrics.make_scorer(metrics.recall_score),
               'f1_score' : metrics.make_scorer(metrics.f1_score)}
    cvres = cross_validate(model, X, y, cv=10,scoring=scoring, verbose = 1,␣
 ↪n_jobs = -1)
    print('Cross validation results by folds, accuracy: {}'.
 ↪format(cvres['test_accuracy']))
    print('Cross validation results by folds, precision: {}'.
 ↪format(cvres['test_precision']))
    print('Cross validation results by folds, recall: {}'.
 ↪format(cvres['test_recall']))
    print('Cross validation results by folds, F1: {}'.
 ↪format(cvres['test_f1_score']))
    print('Accuracy mean: {}'.format(np.mean(cvres['test_accuracy'])))
    print('Precision mean: {}'.format(np.mean(cvres['test_precision'])))
    print('Recall mean: {}'.format(np.mean(cvres['test_recall'])))
    print('F1 mean: {}'.format(np.mean(cvres['test_f1_score'])))
```

[6]:
```
def analyzeResults(x_test, y_test, predictions):
    score = model.score(x_test, y_test)
    print('Accuracy: {}'.format(score))
    cm = metrics.confusion_matrix(y_test, predictions)
    print('Confusion matrix (x = Predicted, y = Actual):')
    print(cm)
    print('Precision : {}'. format(metrics.precision_score(y_test, predictions)))
```

```
    print('Recall : {}'. format(metrics.recall_score(y_test, predictions)))
    print('F1 : {}'. format(metrics.f1_score(y_test, predictions)))
```

[18]:
```
def plotSignificantPixels(model, imgs, imgsBackup):
    imgs = imgsBackup
    abscoef = np.abs(model.coef_[0][1:])
    imgs = imgs.reshape(1250,15,64,64)
    pixels = np.where(abscoef > 0.01)
    for patient in imgs:
        patient = patient.reshape(15 * 64 * 64)
        patient[pixels] = 255
        patient = patient.reshape(15, 64, 64)
    fig=plt.figure(figsize=(15, 15))
    columns = 3
    rows = 5
    for i in range(1, columns*rows +1):
        img = imgs[9][i - 1]
        fig.add_subplot(rows, columns, i)
        plt.imshow(img)
    plt.savefig('slika.pdf')
```

[8]:
```
if __name__ == '__main__':
    X,y,imgs = loadData()
    imgsBackup = imgs
    max_abs_scaler = preprocessing.MaxAbsScaler()
    X = max_abs_scaler.fit_transform(X)
```

```
Loading data...
Loaded 100 images
Loaded 200 images
Loaded 300 images
Loaded 400 images
Loaded 500 images
Loaded 600 images
Loaded 700 images
Loaded 800 images
Loaded 900 images
Loaded 1000 images
Loaded 1100 images
Loaded 1200 images
1250 patients loaded
```

[9]:
```
model = train(X, y)
```

```
Training...

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent
workers.
```

```
max_iter reached after 14 seconds
Done with training

[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   13.9s finished
```

[10]:
```
validate(model,X,y)
```

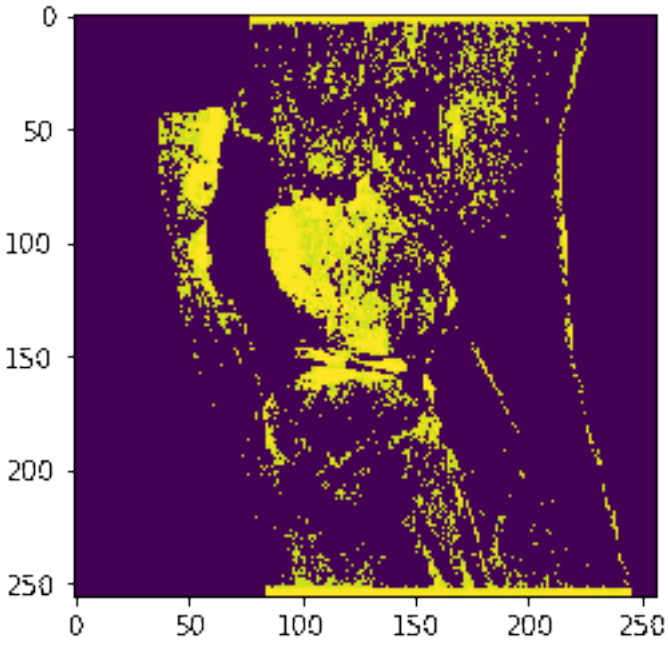
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.

Cross validation results by folds, accuracy: [0.65873016 0.664      0.664
0.736      0.656      0.624
 0.64       0.696      0.624      0.62903226]
Cross validation results by folds, precision: [0.52777778 0.54545455 0.5483871
0.75       0.53333333 0.46875
 0.5        0.62962963 0.47058824 0.47368421]
Cross validation results by folds, recall: [0.42222222 0.4        0.37777778 0.4
0.35555556 0.33333333
 0.31111111 0.37777778 0.35555556 0.40909091]
Cross validation results by folds, F1: [0.4691358  0.46153846 0.44736842
0.52173913 0.42666667 0.38961039
 0.38356164 0.47222222 0.40506329 0.43902439]
Accuracy mean: 0.6591762416794673
Precision mean: 0.5447604828789914
Recall mean: 0.37424242424242427
F1 mean: 0.44159304192130494

[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   17.4s finished
```

[11]:
```
predictions = predict(model,X)
analyzeResults(X,y, predictions)
```

```
Predicting...
Accuracy: 0.9952
Confusion matrix (x = Predicted, y = Actual):
[[801   0]
 [  6 443]]
Precision : 1.0
Recall : 0.9866369710467706
F1 : 0.9932735426008968
```

[24]:
```
plotSignificantPixels(model, imgs, imgsBackup)
```

[20]:
```
with open('input15/0000.pck', 'rb') as file_handler: # originalna segmentirana slika
            volumetric_data = pickle.load(file_handler)
            plt.imshow(volumetric_data[12])
```


0
50
100
150
200
250
0
50
100
150
200
250

[ ]:

Sažetak

U ovom radu će kroz primjere biti predstavljeni neki od problema strojnog učenja te klasične metode za njihovo rješavanje: linearna, logistička i softmax regresija. Objasnit ćemo matematičku pozadinu svake od metoda, upoznat ćemo se sa njihovim nedostatcima te ćemo predstaviti neke od načina korigiranja istih: redukciju broja svojstava i regularizaciju. Pokazat ćemo i primjenu logističke regresije kao modela za klasifikaciju slika iz baze MRNet u praktičnom dijelu rada koji je izrađen u programskom jeziku Python.

Ključne riječi

Linearna regresija, logistička regresija, softmax regresija, redukcija broja svojstava, regularizacija, rijetka matrica, scikit-learn, MRNet, Python.

Abstract

In this paper we will present some of the examples of machine learning problems and the classical methods used for their solving: linear, logistic and softmax regression. We will explain the mathematical background of each of the methods, familiarize ourselves with their shortcomings and present some ways of correcting them, such as reduction of the number of features and regularization. We will also show how to use logistic regression as a classifying model in a practical project made in Python programming language.

Keywords

Linear regression, logistic regression, softmax regression, reduction of the number of features, regularization, sparse matrix, scikit-learn, MRNet, Python.

Životopis

Renato Dean rođen je 25. rujna 1995. godine u Zagrebu. 2010. godine završava Osnovnu školu Ivana Filipovića u Osijeku te nakon toga upisuje III. gimnaziju u Osijeku koju završava 2014. godine. Tijekom svog školskog obrazovanja sudjeluje na brojnim županijskim natjecanjima iz matematike i informatike. Nakon završetka srednje škole, upisuje preddiplomski studij matematike na Odjelu za matematiku u Osijeku te na njemu 2017. godine diplomira temom *Algoritamski pristupi rješavanja Rubikove kocke i implementacija Old Pochmann metode* pod mentorstvom izv.prof.dr.sc Domagoja Matijevića. U jesen 2017. godine upisuje diplomski studij matematike i računarstva na Odjelu za matematiku u Osijeku. Tijekom studija sudjeluje na međunarodnim natjecanjima iz programiranja IEEExtreme te CERC-u. Držao je i demonstrature iz nekoliko kolegija: Uvoda u računarstvo, Uvoda u teoriju brojeva, Dizajniranja i modeliranja baza podataka, Modernih sustava baza podataka te Kombinatorne i diskretne matematike. Na posljednjoj godini diplomskog studija prima nagradu za uspješnost u studiranju te postaje stipendist tvrtke Atos.