Logistička i softmaks regresija (opis metoda, implementacija, primena nad podacima, komparativna analiza) i njihova primena kao aktivacione funkcije u neuronskim mrežama

Student: Danilo Veljović, broj indeksa: 1120

Novembar 6, 2020

Sadržaj

Glava 1

Uvod

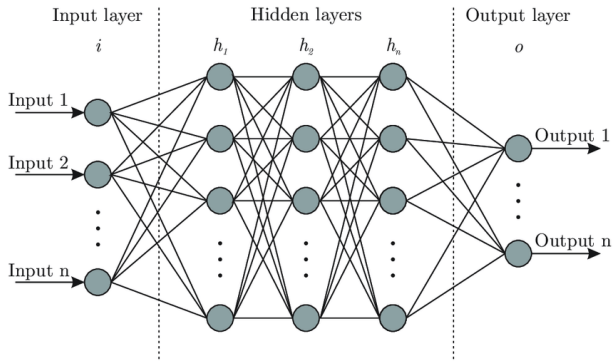
Termini poput veštačke inteligencije i mašinskog učenja su sve popularniji u savremenoj kulturi. Preko medija sve češće čujemo nagoveštaje vodećih naučnika iz ove oblasti da će sistemi koji koriste veštačku inteligenciju zameniti ljude na mnogim radnim mestima. Priča se da će prvo nestati najrepetitivniji poslovi poput vozača kamiona, radnika u skladištima, prodavaca i dr. Treba postaviti suštinsko pitanje zašto je to tako. Zašto će prvo rutinski poslovi nestati prvi? Odgovor na to pitanje daje prave smernice kada neko pokušava da shvati šta je to mašinsko učenje.

Mašinsko učenje vuče korene iz statistike. Slobodno rečeno, mašinsko učenje se može shvatiti kao primena statističkih metoda nad velikom količinom podataka, da bi se izvukle nekakve pravilnosti iz podataka. Te pravilnosti uočava model i opisuje ih različitim statističkim terminima. Model se može shvatiti kao parametrizovana funkcija koja može da „uči" iz podataka. Podaci se u mašinskom učenju dele na trening i test skupove. Trening skupovi imaju određene vrednosti za atribute i imaju izlaznu labelu klase kojoj pripadaju ili neke kontinualne vrednosti koju uzimaju. Model uzima jedan po jedan podatak iz trening skupa, ili čitavu grupu, i proba da napravi predikciju koje bi izlazne vrednosti mogle da odgovaraju tim ulazima. Kada vidi koliko njegova procena odudara od vrednosti labele za taj podatak, on se ispravi svoje parametre. Ovaj proces se naziva treniranje modela. Kada model optimizuje svoje parametre, i kada se odradi evaluacija modela, prelazi se na testiranje. Kada se model testira njemu se daju instance podataka koje nikad nije video. Za te instance on sada predviđa izlazne labele.

Posle ovog kratkog objašnjenja procesa mašinskog učenja, lako se može naslutiti odgovor na prethodno postavljeno pitanje o poslovima koje će prvo „progutati" inteligentni sistemi. Uzmimo primer radnika koji radi u skladištu i pomera kutije od mesta A do mesta B. Robot može da posmatra radnika koji nosi kutije i da zaključi da radnik na mestu A podigne kutiju, nosi je do mesta B i tu je spušta. To je njegov „trening skup" podataka. Kod ovakvog tipa inteligentih sistema najčešće se koriste konvolucione neuronske mreže za računarski vid, pomoću kojih sistem prima podatke o tome šta se dešava oko njega. Princip ostaje isti, sistem preko kamera, prima ulazne podatke, i ovo služi kao trening skup podataka. Trenira određen model, i kada se završi taj proces on može da izvede zadatak prenosa kutija. Ako se od sistema zahteva da radi nešto novo, slično prethodnom, on će probati to da uradi na jedan način, i u zavisnosti od ishoda „učiće" o tome gde je pogrešio, odnosno optimizovaće model.

Tehnike mašinskog učenja su srž inteligentnih sistema. Neki od važnijih modela mašinskog učenja su:

- *Klasifikacija* predviđa klasu koja odgovara podatku
- *Regresija* predviđa kontinualnu vrednost koja odgovara podatku
- *Klasterizacija* je tehnika nenadgledanog učenja. Za razliku od tehnika nadgledanog učenja, o kojima je do sad bilo reči, kod nenadgledanog učenja nemamo informaciju o tome koji podatak ima kakvu klasu/vrednost. Zadatak klasterizacije je da one podatke koji su „slični" (mera sličnost se definiše kao parametar tehnike) grupiše zajedno.
- *Neuronske mreže* su tehnika koja je donela revoluciju u svet mašinskog učenja i omogućila stvari poput računarskog vida i procesuiranja prirodnih jezika. Ideja za ovu tehniku dobijena je iz ideje ljudskog mozga. Cela neuronska mreža se sastoji iz veštačkih neurona, koji su međusobno povezani. Na slici 1.1 dat je prikaz jedne neuronske mreže. Neuronska mreža se sastoji iz tri sloja, *ulaznog* sloja, mnoštvo *skrivenih* slojeva i *izlaznog sloja*. Ideja kod funkcionisanja neuronske mreže je da početni slojevi prepoznaju jako male delove sistema, npr kod računarskog vida prepoznaju ivice, kod procesuiranja prirodnih jezika prepoznaju pojedina slova. Skriveni slojevi prepoznaju složenije pojave, npr geometrijske figure ili tela, ili reči i rečenice. Na kraju izlazni neuroni se aktiviraju ako je prepoznato nešto. Primera radi, mreža može da se trenira da prepoznaje slike mačaka ili da prepoznaje da li je neka rečenica nosi pozitivna ili negativna osećanja. Ako mreža prepozna sliku mačke, ili ako je određena rečenica ima pozitivna osećanja, izlazni neuron se aktivira.



Slika 1.1: Arhitektura neuronske mreže

Jedna od tehnika koja se koristi za klasifikaciju se naziva *logistička regresija*. Iako u nazivu sadrži reč *regresija* ova tehnika se ne koristi za predviđanje kontinualne vrednosti, već za predviđanje da li neki podatak pripada nekoj klasi ili ne. U nastavku rada biće reči o matematičkim osnovama logističke regresije, o funkciji koja je vrlo slična logističkoj regresiji i vrlo često se koristi u sličnim situacijama - *softmaks regresiji*. Implementiraćemo logističku i softmaks regresiju u programskom jeziku Python, primeniti je nad skupom podataka i evaluirati njihove performanse i napraviti komparativnu analizu obe tehnike. Na kraju rada biće dat prikaz primene logističke i softmaks regresije kao aktivacione funkcije („*threshold functions*") kod neuronskih mreža.

Glava 2

Teorijske osnove

2.1 Kratak pregled potrebnih pojmova iz verovatnoće i statistike

Kako bi se pravilno razumela logistička regresija potrebno je pre toga dobro poznavati njene matematičke osnove. U nastavku je data kratka rekapitulacija potrebnih pojmova iz verovatnoće i statistike koji su potrebni za njeno razumevanje. Da bi se lakše svi pojmovi dati u nastavku, razmatrani su samo u okvirima diskretnih slučajnih promenljivih.

2.1.1 Verovatnoća

Verovatnoća predstavlja odnos između broja ishoda u povoljnih za realizaciju događaja i broja svih mogućih ishoda.

$$P = \frac{broj\ pozitivnih\ ishoda\ događaja}{broj\ svih\ mogućih\ ishoda} \tag{2.1}$$

Primer 1. Primer jednog eksperimenta može biti bacanje novčića. Povoljan ishod je kada se padne glava. Broj svih mogućih ishoda u jednom bacanju je 2 (može se pasti ili pismo ili glava). Kada zamenimo ove vrednosti u prethodnu jednačinu dobijamo:

$$P = \frac{1}{2} = 0.5 \tag{2.2}$$

Iz jednačine se vidi da je verovatnoća da u jednom bacanju novčića dobijemo glavu 0.5, odnosno 50 %.

2.1.2 Šansa

Šansa predstavlja odnos između verovatnoće da se događaj desio i verovatnoće da se događaj nije desio.

$$šansa = \frac{P(događaj\ se\ desio)}{P(događaj\ se\ nije\ desio)} = \frac{p}{1\text{-}p} \tag{2.3}$$

Ako se vratimo na primer 1 bacanja novčića, vidimo da je šansa da se događaj desi, odnosno da se padne glava jednak:

$$šansa(glava) = \frac{0.5}{0.5} = 1, tj.1{:}1 \tag{2.4}$$

Zaključujemo da je šansa da se padne glava ista kao i šansa da se ne padne glava, tj 1 prema 1.

2.1.3 Bernulijeva raspodela

Da bi se izbegao rad sa ishodima nekog eksperimenta (koji nisu uvek numerički) uveden je *slučajne promenljive*. Slučajna promenljiva je funkcija koja preslikava skup ishoda u skup realnih brojeva. Raspodela slučajne promenljive predstavlja verovatnoću da slučajna promenljiva uzme neku od mogućih vrednosti.

Definišemo slučajnu promenljivu X = broj uspeha pri bacanju novčića i $X \sim Bernoulli(p)$. Moguće vrednosti koje slučajna promenljiva može da ima (obeležavamo ih sa x) su 0 ili 1. Nula se dobija ako se pri jednom bacanju nije desio uspešan događaj, odnosno 1 ako se desio uspešan događaj. Obeležavamo to sa x = 0, 1.

Bernulijevu raspodelu je najlakše razumeti kroz Bernulijev eksperiment. Bernulijev eksperiment je eksperiment koji se dešava jednom i koji kao rezultat može da ima uspeh (obično obeležen 1) ili neuspeh (obično obeležen 0) tj ima samo dve moguće posledice. Najbolji primer za to je bacanje novčića. Dve moguće posledice su da se padne ili pismo ili glava. Uzmimo da se uspešnim događajem smatra da se pala glava pri bacanju. Odavde sledi da ako se padne pismo, to smatramo neuspehom. Izvedemo eksperiment jednom i ako se padne glava smatramo da se desio uspešan događaj, ako se padne glava smatramo da se desio neuspešan događaj.

Verovatnoća da se desio uspešan događaj je ujedno i jedini parametar raspodele, i obeležava se sa p.

$$P(X = 1) = p \tag{2.5}$$

Jedan parametar se prosleđuje jer imamo samo jedan eksperiment. Odavde se može primetiti da je Bernulijeva raspodela specijalan slučaj Binomne raspodele za $n = 1$.

Verovatnoća da se desio neuspešan događaj je:

$$P(X = 0) = 1 - p = q \tag{2.6}$$

Za primer fer bacanja novčića, verovatnoća uspeha je:

$$P(X = 1) = 0.5 \tag{2.7}$$

Verovatnoća neuspeha je data u jednačini

$$P(X = 0) = 0.5 \tag{2.8}$$

Matematičko očekivanje (srednja vrednost) se za diskretne slučajne promenljive koje imaju Bernulijevu raspodelu dobija kao:

$$\mu = E[X] = \sum_{x_\mathrm{i} \in \mathcal{X}} x_\mathrm{i} p(X = x_\mathrm{i}) = 1 * p + 0 * q = p \tag{2.9}$$

gde je X slučajna promenljiva, x moguće vrednosti slučajne promenljive i $\mathcal{X}$ je skup vrednosti koje slučajna promenljiva može da ima.

Varijansa (mera koja pokazuje meru odstupanja od srednje vrednosti) se dobija kao:

$$\begin{aligned} Var[X] = \sigma^2 = \\ &= Cov[X, X] = \\ &= E[(X - \mu)(X - \mu)] = E[(X - \mu)^2] = \\ &= \sum_{x_\mathrm{i} \in \mathcal{X}}^{\mathcal{X}} (x_\mathrm{i} - \mu)^2 p(X = x_\mathrm{i}) = \\ &= (0 - p)^2 * q + (1 - p)^2 * p = p^2 * q + q^2 * p = pq(p + q) = pq \end{aligned} \tag{2.10}$$

Standardna devijacija predstavlja koren iz varijanse:

$$\sigma = \sqrt{(Var[X])} = \sqrt{pq} \tag{2.11}$$

2.2 Logistička regresija

Logistička regresija je tehnika koja se koristi za klasifikaciju. Najčešće primenu nalazi u binarnoj klasifikaciji. Pomoću nje se određuje da li neki podatak pripada nekoj klasi ili ne. Da bi se neki skup podataka mogao modelirati logističkom regresijom, on mora imati Bernulijevu raspodelu. U skladu s tim se očekuje da se na izlazu mogu pojaviti samo dve vrednosti. Nula na izlazu znači da podatak ne pripada klasi, dok jedinica znači da podatak pripada klasi.

2.2.1 Procena verovatnoće

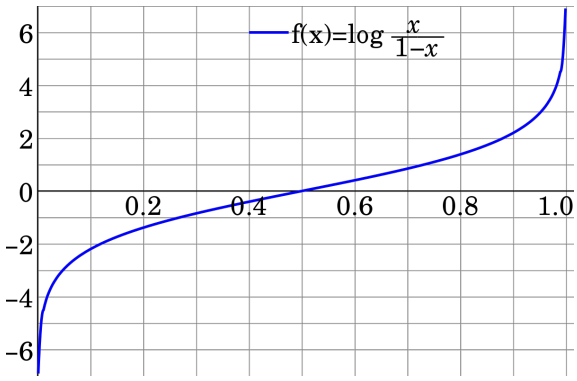
Slično linearnoj regresiji, logistička regresija procenjuje nepoznatu verovatnoću linearne kombinacije elemenata ulaznog vektora. Nepoznatu verovatnoću procenjuje logističkom funkcijom. Nakon procene verovatnoće određene linearne kombinacije elemenata ulaznog vektora, mapira tu verovatnoću na 0 ili 1. Ako je procenjena verovatnoće p veća od 0.5 daje izlaz 1, odnosno 0, ako je verovatnoća manja od 0.5. Procenjena verovatnoća p se često obelažava sa $\hat{p}$.

Skup ulaznih podataka kod logističke regresije, bar u slučaju mašinskog učenja, predstavljen je kao vektor $\theta^T \mathbf{x}$, gde je $\mathbf{x} = \mathrm{x}_1, \mathrm{x}_2, ..., \mathrm{x}_\mathrm{n}$ a $\theta = \theta_1, \theta_2, ..., \theta_\mathrm{n}$, odnosno $\mathbf{x}$ je vektor atributa podataka na ulazu, a θ su odgovarajuće težine svakog od atributa.

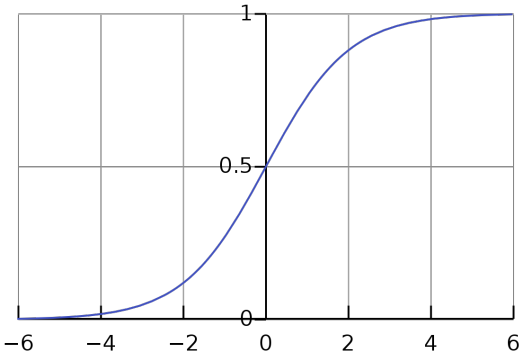
Potrebno je naći vezu između nezavisnih promenljivih na ulazu i izlaza tako da, kao kod Bernulijeve raspodele, izlaz bude ili 0 ili 1. Ta veza se predstavlja *logit* funkcijom. Logit funkcija mapira linearnu kombinaciju nezavisno promenljivih na ulazu na izlaz . Logit funkcija predstavlja prirodni logaritam šanse. Logit funkcija je podobna jer ima samo jedan ulazni parametar koji ce kasnije biti verovatnoća p

$$\mathit{logit(p)} = ln\left(\frac{p}{\mathit{1\text{-}p}}\right) \tag{2.12}$$

Logit funkcija je data na slici 2.1. Domen funkcije je interval $(0, +1)$, i definisana je svuda sem u 0 i 1. Kodomen funkcije je interval $(-\infty, +\infty)$. Pošto je potrebno da nama argumenti mogu da uzmu bilo koju realnu vrednost, a da izlaz bude u intervalu između 0 i 1, potražićemo inverznu funkciju ovoj. Ta funkcija se naziva *logistička*



Slika 2.1: Logit funkcija



Slika 2.2: Logistička funkcija

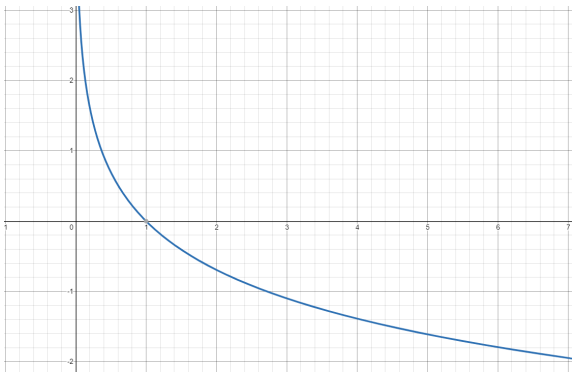
funkcija. Izvođenje je dato u jednačini 2.12. Finalni oblik logističke funkcije dat je u jednačini 2.13.

$$\begin{aligned} ln\left(\frac{p}{1\text{-}p}\right) &= \alpha, \alpha \in \mathbb{R} \\ \frac{p}{1-p} &= e^{\alpha} \\ p &= e^{\alpha}(1-p) \\ p + pe^{\alpha} - e^{\alpha} &= 0 \\ p(1+e^{\alpha}) - e^{\alpha} &= 0 \\ p &= \frac{e^{\alpha}}{1+e^{\alpha}} \end{aligned} \tag{2.13}$$

$$logit^{-1}(\alpha) = h_{\theta}(\alpha) = \sigma(\alpha) = \frac{e^{\alpha}}{1+e^{\alpha}} = \frac{1}{1+e^{-\alpha}} \tag{2.14}$$

Grafik logističke funkcije dat je na slici 2.2. Vidimo da je sada domen funkcije $(-\infty, +\infty)$, a kodomen $(0, +1)$. Ovakva funkcija se naziva i *sigmoidnom* funkcijom (takođe postoji naziv i „S"-kriva) i ona daje broj uvek između 0 i 1.
Kada logistička regresija nađe verovatnoću $\hat{p}$ kao funkciju ulaznih parametara , ona na osnovu te verovatnoće može da da predikciju da li taj podatak pripada pozitivnoj klasi.

$$\hat{y} = \begin{cases} 1, & \hat{p} > 0.5 \\ 0, & \hat{p} < 0.5 \end{cases} \tag{2.15}$$



Slika 2.3: Grafik funkcije -ln(x)

2.2.2 Treniranje modela i funkcija gubitka

Cilj treniranja modela je da se podese parametri vektora θ tako da model daje visoku verovatnoću za pozitivne instance (y = 1) i nisku verovatnoću za negativne instance (y = 0). Funkcija gubitka se koristi kada se optimizuju parametri modela. Funkcija gubitka mapira neku realnu vrednost, najčešće je to nekakva razlika predviđene vrednosti i stvarne vrednosti labele neke instance, u realan broj. Taj realan broj predstavlja cenu. Cilj optimizacije modela je da minimizira funkciju gubitka. Početni izgled funkcije gubitka za logističku regresiju je dat u jednačini 2.16.

$$c(\theta) = \begin{cases} -ln(\hat{\text{p}}), & y = 1 \\ -ln(1-\hat{\text{p}}), & y = 0 \end{cases} \tag{2.16}$$

Na slici 2.3 je dat grafik funkcije $-ln(x)$. Kada $x \to 0$, tada $-ln(x) \to \infty$. Zbog toga će cena biti velika ako model proceni verovatnoću koja teži nuli za pozitivne instance. Takođe biće jako velika ako model proceni verovatnoću blizu jedinice za negativne instance. Ako je x u okolini jedinice, tada će -ln(x) imati vrednost oko nule. Odnosno ako cena će biti 0 ako je procenjena verovatnoća blizu 0 za negativne instance, ili blizu 1 za pozitivne instance. Funkcija 2.16 se može takođe napisati kao:

$$c(\theta) = [\ yln(\hat{p}) + (1-y)ln(1-\hat{p})] \tag{2.17}$$

Odnosno za sve elemente ulazne matrice dobijamo konačni oblik, dat u jednačini 2.18.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[\ y^i ln(\hat{p}^i) + (1-y^i)ln(1-\hat{p}^i)] \tag{2.18}$$

Konstanta $\frac{1}{m}$ se koristi da bi se našla prosečna vrednost svih grešaka. Jednačina zatvorenog oblika koja izračunava optimalne vrednosti θ vektora nije poznata. Međutim ova funkcija jeste konveksna, tako da se optimizacionim algoritmom opadajućeg gradijenta može naći globalni minimum funkcije. Za globalni minimum funkcije gubitaka svi parametri vektora θ imaju optimalne vrednosti, odnosno najmanja je razlika između pravih labela i procenjenih labela. Kada se stigne do globalnog minimuma nije više moguće smanjivati razliku između procenjene i realne vrednosti.

U nastavku biće dat postupak dobijanja gradijenta funkcije 2.18. Biće dat postupak dobijanja parcijalnog izvoda po promenljivoj θ_{j}. Gradijent funkcije je specijalan slučaj Jakobijeve matrice, kada je funkcija skalarna. U opštem slučaju gradijent skalarne diferencijabilne funkcije više promenljih je vektorsko polje ∇f čija vrednost u tački

p je vektor čije komponente su parcijalni izvodi f u p. Odnosno za f: $\mathbb{R}^n \to \mathbb{R}$, njen gradijent je definisan kao: ∇f: $\mathbb{R}^n \to \mathbb{R}^n$ je definisan u tački $p = (x_1, x_2, ..., x_n)$ u n-to dimenzionalnom prostoru kao vektor:

$$\nabla f(p) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \\ \frac{\partial f}{\partial x_2} \\ . \\ . \\ . \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{2.19}$$

Gradijent vektor se može protumačiti kao „pravac i stopa najbržeg rasta" funkcije. Ako je gradijent funkcije u tački p nenulti, pravac gradijenta je pravac u kome funkcija najbrže raste od tačke p. Intenzitet gradijenta je stopa rasta u tom pravcu. Gradijent je nulti vektor u tački samo ako je to stacionarna tačka, tj. ako je to tačka ekstremuma. Kasnije će biti potrebno da iterativnim metodama smanjujemo vrednost funkcije i za ovo će nam biti potreban negativan gradijent funkcije, umesto pozitivnog. Funkcija koja iterativno računa globalni minimum, metodom opadajućeg gradijenta je data na slici 2.20

$$a_{n+1} = a_n - \gamma \nabla f(a_n) \tag{2.20}$$

Tražimo gradijent funkcije u datoj tački i onda idemo u smeru opadajućeg gradijenta. Vrednost a je trenutna tačka odakle se krećemo, f(n) je funkcija čiji se minimum traži. U ovom slučaju funkcija čiji se minimum traži biće funkcija gubitaka data u jednačini 2.18.

Izvod logističke funkcije

U nastavku je dat izvod logističke funkcije $\sigma(x) = \frac{1}{1+e^{-x}}$.

$$\begin{aligned} \frac{\partial(\sigma(x))}{\partial x} &= \frac{0 * (1 + e^{-x}) - (1) * (e^{-x} * (-1))}{(1 + e^{-x})^2} \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1 - 1 + e^{-x}}{(1 + e^{-x})^2} = \frac{1 + e^{-x}}{(1 + e^{-x})^2} - \frac{1}{(1 + e^{-x})^2} \\ &= \frac{1}{1 + e^{-x}} * \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x)(1 - \sigma(x)) \end{aligned} \tag{2.21}$$

Izvod funkcije gubitaka

U nastavku je dat izvod funkcije gubitaka 2.18.

Korak 1: Nalaženje diferencijala složene funkcije:

$$\begin{aligned}\frac{\partial(J(\theta))}{\partial\theta_{\mathrm{j}}} &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i * \frac{1}{h_\theta(x^i)} * \frac{\partial(h_\theta(x^i)}{\partial\theta_{\mathrm{j}}}\right] + \sum_{i=1}^{m}\left[\ (1-y^i) * \frac{1}{1-h_\theta(x^i)} * \frac{\partial(1-h_\theta(x^i)}{\partial\theta_{\mathrm{j}}}\right]\ \right) \\ &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i * \frac{1}{h_\theta(x^i)} * \sigma(z)(1-\sigma(z)) * \frac{\partial(\theta^T x)}{\partial\theta_{\mathrm{j}}}\right]\right) \\ &\quad -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ (1-y^i) * \frac{1}{1-h_\theta(x^i)} * -\sigma(z)(1-\sigma(z)) * \frac{\partial(\theta^T x)}{\partial\theta_{\mathrm{j}}}\right]\ \right)\end{aligned} \tag{2.22}$$

Korak 2: Zamena sigmoidne funkcije i nalaženje diferencijala argumenta:

$$\begin{aligned}\frac{\partial(J(\theta))}{\partial\theta_{\mathrm{j}}} &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i * \frac{1}{h_\theta(x^i)} * \sigma(z)(1-\sigma(z)) * \frac{\partial(\theta^T x)}{\partial\theta_{\mathrm{j}}}\right]\right) \\ &\quad -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ (1-y^i) * \frac{1}{1-h_\theta(x^i)} * -\sigma(z)(1-\sigma(z)) * \frac{\partial(\theta^T x)}{\partial\theta_{\mathrm{j}}}\right]\ \right) \\ &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i * \frac{1}{h_\theta(x^i)} * h_\theta(x^i)(1-h_\theta(x^i)) * {x_{\mathrm{j}}}^i\right]\right) \\ &\quad -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ (1-y^i) * \frac{1}{1-h_\theta(x^i)} * -h_\theta(x^i)(1-h_\theta(x^i)) * {x_{\mathrm{j}}}^i\right]\ \right)\end{aligned} \tag{2.23}$$

Napomena: z = $\theta^T\mathbf{x}$

Korak 3: Uprošćavanje množenjem:

$$\begin{aligned}\frac{\partial(J(\theta))}{\partial\theta_{\mathrm{j}}} &= \\ &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i * (1-h_\theta(x^i)) * {x_{\mathrm{j}}}^i - (1-y^i) * h_\theta(x^i) * {x_{\mathrm{j}}}^i\right]\ \right) \\ &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i - y^i * h_\theta(x^i) - h_\theta(x^i) + y^i * h_\theta(x^i)\right]\ {x_{\mathrm{j}}}^i\right) \\ &= -\frac{1}{m}\left(\sum_{i=1}^{m}\left[\ y^i - h_\theta(x^i)\right]\ {x_{\mathrm{j}}}^i\right)\end{aligned} \tag{2.24}$$

Korak 4: Formiranje gradijenta:

$$\nabla f(p) = \frac{\partial(J(\theta))}{\partial\theta} = \begin{pmatrix}\frac{\partial(J(\theta))}{\partial\theta_1}\\ \\ \frac{\partial(J(\theta))}{\partial\theta_2}\\ .\\ .\\ .\\ \frac{\partial(J(\theta))}{\partial\theta_{\mathrm{n}}}\end{pmatrix} = \frac{1}{m}X^T[h_\theta - y] \tag{2.25}$$

Gde je X matrica čiji su redovi instance trening skupa modela.

2.3 Softmaks regresija

Logistička regresija se može generalizovati tako da podržava višeklasnu klasifikaciju, bez potrebe da se trenira i kombinuje više binarnih klasifikatora. Ovakav tip regresije se naziva *softmax regresija*, ili *multinomna logistička regresija*.

Kada se softmaks regresiji prosledi instanca $\mathbf{x}$, prvo se za tu instancu izračunava rezultat funkcije $s_{\mathrm{k}}(x)$ za svaku klasu k. Zatim se procenjuje verovatnoća za svaku klasu primenom *softmaks funkcije* nad vrednostima $s_{\mathrm{k}}(x)$. Jednačina za izračunavanje vrednosti $s_{\mathrm{k}}(x)$ je data na slici 2.26

$$s_{\mathrm{k}}(x) = \mathbf{x}^T \theta^k \tag{2.26}$$

U jednačini se može primetiti da svaka klasa ima svoj poseban vektorski parametar θ^k. Svi ovi vektori se pamte kao redovi u matrici parametara Θ. Kada se izračuna vrednost za instancu $\mathbf{x}$ može se proceniti verovatnoća $\hat{p}_{\mathrm{k}}$ da instanca pripada klasi k. To se radi tako što se vrednosti $s_{\mathrm{k}}(x)$ prosleđuju kao parametri softmaks funkciji (slika 2.27). Softmaks funkcija računa $e^{s_{\mathrm{k}}(x)}$ za svaku vrednost $s_{\mathrm{k}}(x)$ koju jedna instanca ima. Zatim svaku od vrednosti normalizuje. Ako imamo k klasa po kojima vršimo klasifikaciju, rezultat koji se dobija je k verovatnoća, po jedna za svaku klasu kojoj se vrši klasifikacija, za svaku instancu u skupu podataka.

$$\hat{p}_{\mathrm{k}} = \sigma(\mathbf{s(x)})_{\mathrm{k}} = \frac{e^{s_{\mathrm{k}}(x)}}{\sum_{j=1}^{K} s_{\mathrm{j}}(x)} \tag{2.27}$$

Oznake u jednačini 2.27: K broj klasa po kojima se vrši klasifikacija, $\mathbf{s(x)}$ je vektor koji sadrži softmaks rezultate za svaku klasu, a $\sigma(\mathbf{s(x)})_{\mathrm{k}}$ je procenjena verovatnoća da ta instanca $\mathbf{x}$ pripada klasi k

Kao kod logističke regresije, i softmaks regresija kao izlaz će imati oznaku klase koja ima najveću verovatnoću za tu instancu kao što se vidi u jednačini 2.28.

$$\hat{y}_{\mathrm{k}} = \underset{k}{\operatorname{argmax}}\, \sigma(\mathbf{s(x)})_{\mathrm{k}} = \underset{k}{\operatorname{argmax}}\, s_{\mathrm{k}}(\mathbf{x}) = \underset{k}{\operatorname{argmax}}((\theta^k)^T)\mathbf{x}) \tag{2.28}$$

Operator *argmax* vraća vrednost promenljive koja maksimizuje funkciju. U ovoj jednačini vraća vrednost k koja maksimizuje procenjenu verovatnoću $\sigma(\mathbf{s(x)})_{\mathrm{k}}$. Softmaks regresija vraća jednu klasu po instanci i stoga bi je trebalo koristi samo sa klasama koje su međusobno isključive.

Cilj treniranja modela je da on daje visoku verovatnoću za klase kojima trening instance pripadaju i nisku verovatnoću za sve ostale klase. Minimizacija funkcije gubitaka (koja se još naziva i *cross-entropy*) će dati takav rezultat. Ona „kažnjava" model kada da nisku verovatnoću za klasu kojoj trening instanca pripada. Cross-entropy funkcija se često koristi kada se meri koliko dobro skup procenjenih verovatnoća odgovara skupu klasa kojima instance stvarno pripadaju.

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} {y_{\mathrm{k}}}^{(i)} ln({\hat{p}_{\mathrm{k}}}^{(i)}) \tag{2.29}$$

Oznake u jednačini 2.29: ${y_{\mathrm{k}}}^{(i)}$ je verovatnoća da li i-ta instanca pripada klasi k. U praksi generalno može biti jednaka 1 ili 0 jer instanca može samo da pripada ili ne

pripada, nema delimičnog pripadanja. Za K = 2 (kada postoje samo dve klase), ova funkcija gubitka je jednaka funkciji gubitka kod logističke regresije.

Gradijent vektor ove funkcije gubitaka po θ^k je dat u jednačini 2.30.

$$\nabla_{\theta^k} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} (\hat{p}_{\mathrm{k}}{}^{(i)} - y_{\mathrm{k}}{}^{(i)}) \mathbf{x}^i \tag{2.30}$$

Sad se može izračunati gradijent vektor za svaku klasu, onda da se iskoristi metoda opdajućeg gradijenta da se nađu parametri u matrixi Θ koji minimizuju funkciju gubitaka.

Glava 3

Implementacija

3.1 Logistička regresija

U nastavku biće data kompletna implementacija logističke regresije u programskom jeziku Python. Uz programski jezik Python, korišćena je i biblioteka NumPy za numerička izračunavanja i Pandas biblioteka za lakšu manipulaciju podacima. Python je slabo tipiziran jezik koji ima dinamičku proveru tipova i podršku za objektno orijentisanu paradigmu. Uz to takođe ima veliku podršku za biblioteke otvorenog koda. Sama implementacija je rađena u Jupyter svesci.

3.1.1 Opis klase

Funkcionalnosti logističke regresije su enkapsulirane u klasi `LogisticRegression`. Objekti klase se kreiraju pomoću konstruktora kome se prosleđuje broj interacija i parametar α koji predstavlja brzinu učenja (engl. *learning rate*). Brzina učenja je parametar u funkciji optimizacije (u ovom slučaju algoritma opadajućeg gradijenta) koji govori koliko brzo trenutno rešenje konvergira minimumu funkcije. Najčešće se za taj parametar bira vrednost 10^{-3} pa se koriguje u zavisnosti od rezultata. Na slici 3.1 je data implementacija konstruktora klase `LogisticRegression`.

```
class LogisticRegression:
    def __init__(self, lr = 0.01, num_iter = 10000, fit_intercept = True, verbose = False):
        self.lr = lr
        self.num_inter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose
```

Slika 3.1: Konstruktor klase `LogisticRegression`

Funkcije koje enkapsulira klasa su sigmoidna funkcija (datu u jednačini 2.14) koja daje predviđenu verovatnoću da instanca pripada pozitivnoj klasi i funkcija gubitaka koja smanjuje razliku između predviđene klase i klase kojoj trening instanca zaista pripada (datu u jednačini 2.18). Delovi koda koji prikazuju implementaciju dati su na slici 3.2.

Jedna od funkcija koja je javno dostupna je funkcija `fit`. Ova funkcija je kreirana po ugledu na funkcije iz biblioteke otvorenog koda `scikit-learn`. Ona služi za treniranje modela. Kao parametre ima matricu podataka obeleženu sa $\mathbf{X}$, gde jedna vrsta matrice predstavlja jednu instancu, i vektor $\mathbf{y}$ koji sadrži labele za svaku instancu. Funkcija u okviru treniranja modela vrši predviđanja, nalazi gradijent

```
def __sigmoid(self, z):
    return 1/(1 + np.exp(-z))

def __loss(self, h, y):
    return(-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

Slika 3.2: Implementacija sigmoidne funkcije i funkcije gubitaka klase `LogisticRegression`

funkcije u svakoj tački i onda optimizuje parametre θ. To ponavlja određeni broj iteracija. Što je broj iteracija veći to će predviđanja biti preciznija, jer se će funkcija u svakoj iteraciji kretati bliže globalnom minimumu. Implementacija funkcije koja trenira model je data na slici 3.3.

```
def fit(self, X, y):
    if(self.fit_intercept):
        X = self.__add_intercept(X)

    self.theta = np.zeros(X.shape[1])

    for i in range(self.num_inter):
        z = np.dot(X, self.theta)
        h = self.__sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / y.size
        self.theta -= self.lr * gradient

        if(self.verbose == True and i % 10000 == 0):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            print(f'loss: {self.__loss(h, y)} \t')
```

Slika 3.3: Implementacija funkcije za treniranje modela klase `LogisticRegression`

Nakon treniranja modela potrebno je da model može da vrši predviđanja za instance čije su labele nepoznate. Ovo se postiže funkcijama `predict prob` i `predict`. Obe klase imaju `public` modifikator pristupa što znači da bilo ko može da im pristupi. Funkcija `predict prob` uzima jednu instancu (vektor) ili više instanci (matricu) i vraća predviđenu verovatnoću da instanca/instance pripada/ju pozitivnoj klasi. Funkcija `predict` takođe uzima kao parametar jednu instancu (vektor) ili više instanci (matricu) i vraća klasu koju je model predvideo da ta instanca/e pripada/ju. Implementacija funkcija `predict prob` i `predict` je data na slici 3.4.

```
def predict_prob(self, X):
    if self.fit_intercept:
        X = self.__add_intercept(X)

    return self.__sigmoid(np.dot(X, self.theta))

def predict(self, X):
    return self.predict_prob(X).round()
```

Slika 3.4: Implementacije funkcija `predict prob` i `predict` klase `LogisticRegression`

3.1.2 Primena i analiza rezultata

U ovom delu ćemo implementirati projekat u kome ćemo klasu `LogisticRegression` za predikciju podataka. Glavni koraci pri izradi jednog projekta su:

1. Pribavljanje podataka
2. Vizualizacija i upoznavanje podataka
3. Priprema podataka za obradu
4. Odabir modela i treniranje
5. Evaluacija modela
6. Testiranje modela

Pribavljanje i vizualizacija podataka

Najčešći izvori podataka za analizu su relacione baze podataka. Vrlo često je potrebno da se podaci koji se nalaze u različitim tabelama denormalizuju i onda povežu u jednu logičku tabelu. U ovom projektu neće biti korišćeni podaci koji se nalaze u bazi podataka, već podaci u .csv formatu. Skup podataka koji će se koristiti je skup podataka o biljci perunici. Ovaj skup podataka od atributa sadrži dužinu i širinu čašičnog listića, kao i dužinu i širinu latice ove biljke. Uz ova četiri atributa takođe sadrži i informacije o vrsti perunike, odnosno labelu klase kojoj instanca pripada. Tri moguće klase, odnosno vrste perunike, kojima instance mogu da pripadaju su: *Iris setosa, Iris virginica i Iris versicolor*. Skup podataka sadrži po 50 instanci svake klase.

Ovaj dataset je sastavni deo biblioteke scikit-learn i kod za njegovo učitavanje je dat na slici 3.5. Na slici se vidi da je matrica **X** dimenzija 150x4, odnosno 150

```
import sklearn.datasets

iris = sklearn.datasets.load_iris()
X = iris.data
y = (iris.target) * 1

print(X.shape, y.shape)
```

```
(150, 4) (150,)
```

Slika 3.5: Učitavanje skupa podataka za perunike

instanci sa po četiri atributa. Vektor y sadrži 150 labela koje su oznake klase kojoj svaka instanca pripada.

Kako bi se stekao nekakav intuitivni osećaj za podatke, moguće je da se tabelarno prikaže prvih 5 elemenata skupa podataka. Ovo se u Pythonu postiže naredbom `head()`. Efekti naredbe su dati na slici 3.6. Sa slike se može primetiti da su svi podaci mereni u cm.

```
In [120]: df = pd.DataFrame(iris.data, columns=iris.feature_names)
          df.head()
```

Out[120]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Slika 3.6: Prvih 5 elemenata skupa podataka

Naredbom `describe()` moguće je dobiti statističke podatke o skupu podataka. Funkcija vraća tabelu gde svaka kolona odgovara jednom atributu. U redovima tabele se nalaze informacije o broju instanci koje imaju ovaj podatak (gde je ova vrednost različita od null), o matematičkom očekivanju za taj atribut, o standardnoj devijaciji, o minimalnom i maksimalnom elementu skupa itd. Rezultat naredbe vidljiv je na slici 3.7.

```
In [134]: df.describe()
```

Out[134]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Slika 3.7: Primena `describe` metode nad skupom podataka

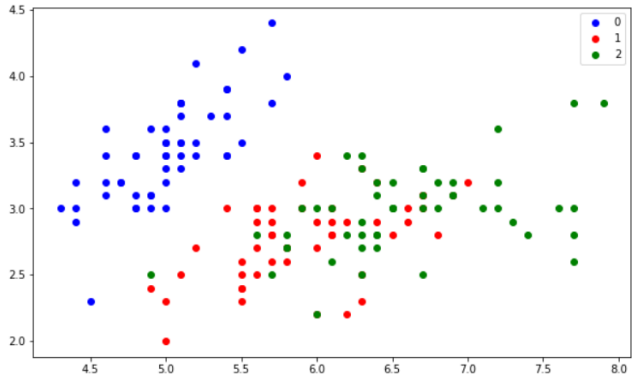
Vizualizacija podataka je jako važan korak pri upoznavanju skupa podataka. U Pythonu ovo se postiže funkcijama iz biblioteke otvorenog koda *matplotlib*. Na grafiku na slici 3.8 na x osi imamo predstavljenu dužinu čašičnog listića, a na y osi širinu čašičnog listića. Instance su dobile boju na osnovu klase kojoj pripadaju. Vidimo da ako gledamo samo na osnovu ova dva atributa, klasa 1 čini jednu grupu, a klase 2 i 3 drugu grupu. Drugim rečima klase 1 i klase 2 i 3 su linearno razdvojive. Ovo je dobar problem za binarnu klasifikaciju i samim tim testiranje malopre implementirane klase za logističku regresiju. Model će biti treniran da prepoznaje da li instance pripadaju klasi 1 ili ne pripadaju.

Priprema podataka i treniranje modela

Pre nego što je neki skup podataka spreman da se njime trenira i testira model, potrebno je podatke skalirati u neki opseg i rešiti se null vrednosti koje postoje u podacima. Skup podataka koji koristimo u ovom primeru je već prečišćen, odnosno skup podataka nema null vrednosti i sve su vrednosti otprilike istog reda veličine pa ih nema potrebe dalje skalirati. Treniranje modela je proces u kome se modelu, prosleđuje jedna po jedna ili čitava grupa instanci nekog skupa podataka, sa labelama (oznake kojoj klasi pripadaju ili koju kontinualnu vrednost imaju kao labelu). Model ima funkciju kojom vrši predikciju labele za neku instancu. Zatim upoređuje labelu

```
In [119]: import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='1')
plt.scatter(X[y == 2][:, 0], X[y == 2][:, 1], color='g', label='2')
plt.legend();
```



Slika 3.8: Vizuelizacija podataka za skup podataka o perunici

koju je predvideo sa labelom koja odgovara tom modelu. Ako se labele ne poklapaju model nekom metodom optimizacije optimizuje svoje parametre da bolje odgovaraju skupu podataka. Krajnji cilj je minimum funkcije gubitaka koja se optimizuje. Pošto će isti skup podataka biti korišćen i za treniranje i za testiranje, potrebno ga je podeliti u test i trening skup podataka. Trening i test skup podataka se deli u razmeri od 80:20. Trening skup podataka se koristi za treniranje modela, a test skup podataka za testiranje performansi modela. Funkcija koja će podeliti skup podatka u trening i test je data na slici 3.9.

```
In [30]: X_train, X_test, y_train, y_test = X[:100], X[100:], y[:100], y[100:]
         print(X_train.shape, y_train.shape)

         (100, 2) (100,)
```

Slika 3.9: Podela podataka na trening i test skup

Treniranje modela se vrši pozivanjem `fit()` funkcije instance klase `LogisticRegression`. Poziv funkcije za treniranje modela je dat na slici 3.10. Funkciji se prosleduju vrednosti atributa u trening klasi i odgovarajuće labele.

```
In [31]: %time model.fit(X_train, y_train)

         Wall time: 4.56 s
```

Slika 3.10: Podela podataka na trening i test skup

Evaluacija modela

Nakon treniranja modela potrebno je videti kakve performanse daje. Kod klasifikacionih modela glavne performanse mere su *tačnost*,*preciznost*, *odziv*, *matrica konfuzije*, *f-mera* i *ROC kriva*. U nastavku biće dat pregled svake od mera.

Tačnost

Tačnost je mera koja pokazuje koliko je ukupno istanci, od svih mogućih, klasifikator ispravno klasifikovao. To znači koliko je instanci pravilno raspodelio u pozitivnu i negativnu klasu u slučaju ovog binarnog klasifikatora. Da bi se izbegla pristrasnost klasifikatora pri evaluaciji modela najčešće se koristi *k-fold cross validation* metoda. Trening skup podataka, najčešće se on korisiti za validaciju, ova metoda deli na k podskupova. Pri evaluaciji modela ovom metodom, treniranje modela se radi k puta. U svakoj iteraciji jedan od k skupova, (svaki put različiti, tako da na kraju treniranja svi dođu na red) se uzima da bude validacioni skup. Ostalih k-1 su trening skupovi. Kada se model istrenira, na validacionom skupu se proverava tačnost klasifikatora. Implementacija i primena k-fold cross validation funkcije je data na slici 3.11.

```
In [37]: from sklearn.model_selection import StratifiedKFold

skfolds = StratifiedKFold(n_splits=3)

for train_index, test_index in skfolds.split(X_train, y_train):
    #clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train[test_index]
    model.fit(X_train_folds, y_train_folds)
    y_pred = model.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))
```

```
1.0
1.0
0.9696969696969697
```

Slika 3.11: K fold cross validation funkcija u evaluaciji modela

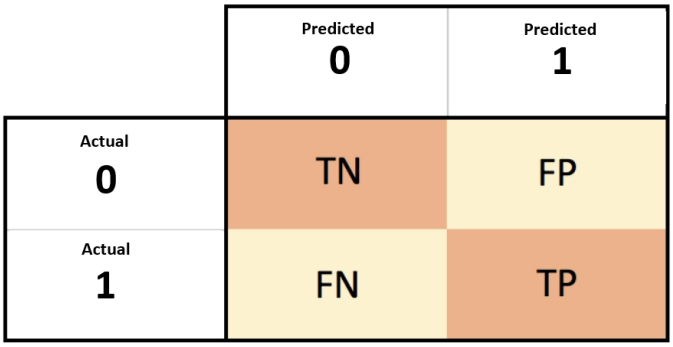
U ovom slučaju se trening skup deli na tri dela i rezultati tačnosti su redom 1.0, 1.0 i 0.97, što je jako dobar rezultat.

Matrica konfuzije

Mnogo temeljniji način za evaluaciju performansi klasifikatora je matrica konfuzije. Matrica konfuzije nam govori koliko je instanci jedne klase klasifikovano kao instanca neke druge klase. Na glavnoj dijagonali se nalazi broj ispravno klasifikovanih instanci, odnosno za binarni klasifikator TP (True Positive - instance koje su klasifikovane kao pozitivne i čija je labela pozitivna, ispravno klasifikovane pozitivne instance) i TN (True Negative - instance koje su klasifikovane kao negativne i čija je labela negativna). Na slici 3.12 je dat šematski prikaz matrice konfuzije za binarni klasifikator. Polje FN (False Negatives) prikazuje broj elemenata čija je predviđena klasa negativna, a koji zapravo pripadaju pozitivnoj klasi. Polje FP (False Positives) prikazuje broj elemenata čija je predviđena klasa pozitivna, a koji zapravo pripadaju negativnoj klasi.

Matrica konfuzije za model logističke regresije primenjen na skup podataka za peruniku je data na slici 3.13.

Zaključak koji se može izvesti iz matrice konfuzije 3.13 je da klasifikator skoro idealno prepoznaje instance klase 1 i skoro da ih uopste ne mesa sa instancama druge klase, sto je i ocekivan rezultat. Rezultat je ocekivan jer smo videli da su instance



Slika 3.12: Primer matrice konfuzije

```
In [36]: from sklearn.metrics import confusion_matrix
         confusion_matrix(ret, rety)

Out[36]: array([[49,  0],
                [ 1, 50]], dtype=int64)
```

Slika 3.13: Matrica konfuzije za model logističke regresije nad skupom podataka vezanim za perunike

klase 1 i ostalih klasa skoro savrseno linearno razdvojive.

Preciznost, odziv i f-mera

Jedna od metrika koja se može dobiti iz matrice konfuzije je preciznost. Preciznost klasifikatora je tačnost pozitivnih predviđanja. Jednačinom 3.1 se računa preciznost klasifikatora.

$$preciznost = \frac{TP}{TP + FP} = \frac{50}{1 + 50} = 0.98 \tag{3.1}$$

Preciznost je 0.67, odnosno od ukupnog broja instanci za koje je klasifikator rekao da su pozitivne, samo 67% je stvarno pozitivno.

$$odziv = \frac{TP}{TP + FN} = \frac{50}{0 + 50} = 1 \tag{3.2}$$

-

Jednačinom 3.2 se računa odziv. Odziv predstavlja odnos između instanci koje je klasifikator prepoznao kao pozitivne i koje stvarno jesu pozitivne i ukupnog broja stvarno pozitivnih instanci u skupu podataka. Ovaj model ima preciznost 1, što znači da je od svih mogućih pozitivnih instanci model prepoznao sve.

$$f - mera = \frac{2}{\frac{1}{preciznost} + \frac{1}{odziv}} = \frac{2}{\frac{1}{0.98} + \frac{1}{1}} = 0.99 \tag{3.3}$$

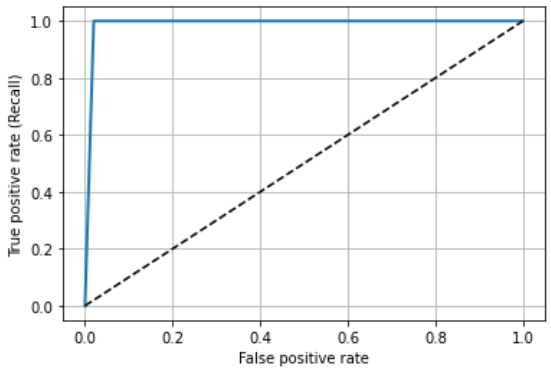
F mera je potrebna kada je poželjno kombinovati odziv i preciznost u jednu meru. F mera je harmonijska sredina preciznosti i odziva (jednačina 3.3). Obično se koristi kada je potreban brz i jednostavan način da se uporede dva klasifikatora. Iz jednačine se vidi da će F mera imati visoku vrednost samo ako su i preciznost i odziv visoki.

ROC kriva

ROC kriva (engl. *Receiver operating characteristic* je još jedan važan alat pri evaluaciji binarnih klasfikatora. Na x-osi se nalazi True Positive Rate (odziv), a na y osi False Positive Rate (odnos negativnih instanci koje su pogrešno klasifikovane kao pozitivne). Idealni klasifikator ima površinu ispod ROC krive jednaku 1. ROC kriva za ovaj klasifikator je data na slici 3.14. Površina ispod krive je data na slici 3.15. Sa slike vidimo da je povrsina ispod krive 0.99 sto znaci da je klasifikator skoro savrsen.

```
In [44]: def plot_roc_curve(fpr, tpr, label=None):
             plt.plot(fpr, tpr, linewidth=2, label=label)
             plt.plot([0,1], [0,1], 'k--')
             plt.axis("on")
             plt.grid()
             plt.xlabel("False positive rate")
             plt.ylabel("True positive rate (Recall)")

         plot_roc_curve(fpr, tpr)
         plt.show()
```



Slika 3.14: ROC kriva

```
In [46]: from sklearn.metrics import roc_auc_score
         roc_auc_score(rety, ret)

Out[46]: 0.99
```

Slika 3.15: Površina ispod ROC krive

Testiranje modela

Testiranje modela se radi tako što se podaci iz dela skupa podataka koji su odvojeni testiranje daju modelu. Model generiše predikcije za te podatke. Zatim računa preciznost klasifikatora na test skupu podataka. Primena testiranja i preciznost klasifikatora su dati na slici 3.16.

```
In [47]: preds = model.predict(X_test)
         (preds == y_test).mean()

Out[47]: 1.0
```

Slika 3.16: Testiranje modela logističke regresije i preciznost

Preciznost klasifikatora nad test podacima je 1, što znači da su sve instance test skupa ispravno klasifikovane.

3.2 Softmaks regresija

U nastavku biće data kompletna implementacija softmaks regresije u programskom jeziku Python. Regresija je implementirana kao klasa `SoftmaxRegression`. Metode koje instanca ove klase može javno pozivati su metoda `fit` i `predict`. Metoda `fit` omogućava treniranje modela softmaks regresije. Metoda `predict` omogućava da se vrši predikcija za podatke čija je klasa nepoznata. Skup podataka koji je korišćen za testiranje modela je takođe *iris* skup podataka. U ovom slučaju smo umesto dve klase koristili sve tri, tako da i za predikciju imamo sve tri vrste klasa.

3.2.1 Opis klase

Funkcionalnosti softmaks regresije su enkapsulirane u klasi `SoftmaxRegression`. Klasa izlaže konstruktor u kome se prosleđuje broj interacija i parametar α koji predstavlja brzinu učenja (engl. *learning rate*). Uz ova dva parametra prosleđuje se i broj klasa koji postoji u skupu podataka, kao parametar K. Na slici 3.17 je dat prikaz konstruktora klase `SoftmaxRegression`.

```
class SoftmaxRegression:
    def __init__(self, K, lr=0.01, num_iter=10000):
        self.lr = lr
        self.num_iter = num_iter
        self.K = K
```

Slika 3.17: Konstruktor klase `SoftmaxRegression`

Funkcije koje enkapsulira klasa su softmaks funkcija (datu u jednačini 2.27) koja daje predviđenu verovatnoću da instanca pripada pozitivnoj klasi i funkcija gubitaka koja smanjuje razliku između predviđene klase i klase kojoj trening instanca zaista pripada (datu u jednačini 2.29). Delovi koda koji prikazuju implementaciju dati su na slici 3.18.

```
def __softmax(self, z):
    z -= np.max(z)
    return np.exp(z) / np.sum(np.exp(z))

def __h(self, X, y):
    return self.__softmax(X @ self.theta)

def __J(self, preds, y, m):
    return np.sum(- np.log(preds[np.arange(m), y]))

def __T(self, y, K):
    # one hot encoding
    one_hot = np.zeros((len(y), K))
    one_hot[np.arange(len(y)), y] = 1
    return one_hot

def __compute_gradient(self, theta, X, y, m):
    preds = self.__h(X, theta)
    gradient = 1 / m * X.T @ (preds - self.__T(y, self.K))
    return gradient
```

Slika 3.18: Implementacija softmaks funkcije i funkcije gubitaka klase `SoftmaxRegression`

Jedna od funkcija koja je izložena spoljnoj upotrebi je funkcija `fit`. Ona služi za treniranje modela. Kao parametre ima matricu podataka obeleženu sa **X**, gde jedna vrsta matrice predstavlja jednu instancu, i vektor **y** koji sadrži labele za svaku instancu. Funkcija u okviru treniranja modela vrši predviđanja, nalazi gradijent funkcije u svakoj tački i onda optimizuje parametre θ. To ponavlja određeni broj iteracija. Što je broj iteracija veći to će predviđanja biti preciznija, jer se će funkcija u svakoj iteraciji kretati bliže globalnom minimumu. Implementacija funkcije koja trenira model je data na slici 3.19.

```
def fit(self, X, y):
    hist = {'loss': [], 'acc': []}
    m, n = X.shape
    np.random.seed(0)
    self.theta = np.random.random((n, self.K))
    for i in range(self.num_iter):
        gradient = self.__compute_gradient(self.theta, X, y, m)
        self.theta -= self.lr * gradient

        # loss
        preds = self.__h(X, self.theta)
        loss = self.__J(preds, y, m)

        c = 0
        for j in range(len(y)):
            if np.argmax(self.__h(X[j], self.theta)) == y[j]:
                c += 1
            acc = c / len(y)
            hist['acc'].append(acc)
        # print stats
        if i % 5000 == 0:
            print('{:.2f} {:.2f}%'.format(loss, acc * 100))
```

Slika 3.19: Implementacija funkcije za treniranje modela klase `SoftmaxRegression`

Nakon treniranja modela potrebno je da model može da vrši predviđanja za instance čije su labele nepoznate. Ovo se postiže funkcijama `predict prob` i `predict`. Obe klase imaju `public` modifikator pristupa što znači da bilo ko može da im pristupi. Funkcija `predict prob` uzima jednu instancu (vektor) ili više instanci (matricu) i vraća predviđenu verovatnoću da instanca/instance pripada/ju pozitivnoj klasi. Funkcija `predict` takođe uzima kao parametar jednu instancu (vektor) ili više instanci (matricu) i vraća klasu koju je model predvideo da ta instanca/e pripada/ju. Implementacija funkcija `predict prob` i `predict` je data na slici 3.20.

```
def predict_prob(self, X):
        return self.__softmax(np.dot(X, self.theta))

def predict(self, X):
        return np.argmax(self.predict_prob(X), axis=1)
```

Slika 3.20: Implementacije funkcija `predict prob` i `predict` klase `SoftmaxRegression`

3.2.2 Primena i analiza rezultata

Softmaks regresija je vrsta višeklasne regresije. Kod višeklasne regresije postoji više klasa za koje se instance klasifikuju. Jedna instanca uvek pripada jednoj klasi.

U ovom delu ćemo implementirati projekat u kome ćemo klasu `SoftmaxRegression` za predikciju podataka. Za treniranje i implementaciju modela koristićemo isti skup podataka o perunikama kao za logističku regresiju. U narednoj sekciji ćemo se kratko osvrnuti na proces treniranja modela, evaluaciju i testiranje modela. Pošto je skup podataka isti deo koji se odnosi na pribavljanje podataka, vizuelizaciju i pripremu podataka biće izostavljen, pošto je već priložen u sekciji 3.1.2.

Treniranje i evaluacija modela

Za treniranje modela podelićemo skup podataka u dva podskupa. Podskup za treniranje će sadržati 140 instanci, a podskup za testiranje će sadržati samo 10 instanci. Podela skupa podataka na trening i test skup data je na slici 3.21.

```
In [118]: X_train, X_test, y_train, y_test = X[:140], X[140:], y[:140], y[140:]
          print(X_train.shape, y_train.shape)

          (140, 5) (140,)
```

Slika 3.21: Podela skupa podataka na trening i test skup za softmaks regresiju

Treniranje modela se radi funkcijom `fit()`. Poziv funkcije za treniranje modela je dat na slici 3.22.

```
In [128]: model.fit(X_train, y_train)

          1060.41 22.14%
          8797.92 82.86%
          17574.56 82.14%
```

Slika 3.22: Podela skupa podataka na trening i test skup za softmaks regresiju

Nakon treniranja modela, za evaluaciju biće prikazane metrike poput matrice konfuzije, preciznosti, odziva i f-mere. Vrednosti za sve ove mere biće generisane preko biblioteke scikit-learn. Kod kojim se dobijaju je dat na slici 3.23.

```
In [142]: from sklearn.metrics import confusion_matrix
          confusion_matrix(ret, rety)

Out[142]: array([[50,  0,  0],
                 [ 0, 28,  2],
                 [ 0, 22, 38]], dtype=int64)

In [157]: from sklearn.metrics import precision_recall_fscore_support as score

          precision, recall, fscore, support = score(rety, ret)

          print('precision: {}'.format(precision))
          print('recall: {}'.format(recall))
          print('fscore: {}'.format(fscore))
          print('support: {}'.format(support))

          precision: [1.         0.93333333 0.63333333]
          recall: [1.   0.56 0.95]
          fscore: [1.   0.7  0.76]
          support: [50 50 40]
```

Slika 3.23: Evaluacija modela za softmaks regresiju

Iz matrice konfuzije se jasno vidi da model savršeno prepoznaje instance prve klase, međutim meša instance koje pripadaju klasi 2 i 3. Ovo je bila činjenica koju

smo eksploatisali kada smo ovaj skup podataka koristili da logističkom regresijom radimo binarnu klasifikaciju za klasu 1. Da bi se izbegla pristrasnost klasifikatora pri evaluaciji modela najčešće se koristi *k-fold cross validation* metoda. Korišćenjem metode `precision recall fscore support()` iz biblioteke sklearn, dobićemo preciznost, odziv f meru za svaku od klasa. Iz priloženih metrika se jasno vidi da model savršeno prepoznaje sve instance klase 1, a da meša instance klase 2 i 3. Preciznost je najlošija za klasu 3, što znači da od svih instanci koje je klasifikator prepoznao kao da pripadaju klasi 3, samo 63% zaista pripada klasi 3. Najlošiji odziv ima klasa 2, što znači da je klasifikator od svih instanci koje pripadaju klasi 2, samo njih 56% prepoznao kao da pripadaju klasi 2.

Testiranje modela

Testiranje modela se radi sa 10 instanci koje su izdvojene iz početnog skupa podataka. Kod kojim se instance prosleđuju istreniranom modelu je dat na slici 3.24

```
In [146]: y_pred = model.predict(X_test)
          print(y_pred)
          print(y_test)

          [2 2 2 2 2 2 2 2 2 2]
          [2 2 2 2 2 2 2 2 2 2]
```

Slika 3.24: Testiranje modela za softmaks regresiju

Modelu su prosleđene instance koje pripadaju klasi 2, i on ih je sve ispravno klasifikovao.

Glava 4

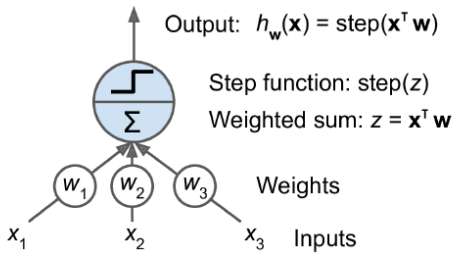
Primena kod neuronskih mreža

Veštačke neuronske mreže su inspirisane idejom bioloških neurona. Vremenom je razlika između prirodnih neurona i veštačkih postajala sve veća, ali se ime i dalje zadržalo. Veštačke neuronske mreže (u daljem tekstu *NN*) su sama srž *dubokog učenja (engl. deep learning)*. NN su moćne, sklabilne i mogu da se primene na veliki broj problema. Neki od problema na koje se mogu primeniti su: klasifikacija velikog broja slika, prepoznavanje govora, preporuka video klipova itd.

4.1 Uvod u veštačke neuronske mreže

Model veštačkog neurona ima jedan ili više binarnih (1/0) ulaza i jedan binarni izlaz. Veštački neuron aktivira svoj izlaz kada je određeni broj ulaza aktivan. Arhitekture sastavljene od ovog modela su bile pojednostavljene ali su mogle da izvršavaju jednostavne logičke operacije poput: $\wedge, \vee, \neg$ itd.

Vremenom se došlo do bolje arhitekture veštačkih neuronskih mreža, tzv. **Perceptron** model. Perceptron arhitektura je smišljena 1957. godine. Zasnovana je na nešto drugačijem modelu veštačkog neurona koji se naziva *threshold logic unit (srp. logička jedinica sa pragom)* u daljem tekstu TLU. Ulazi i izlazi u TLU su brojevi (ne binarne 1/0 vrednosti) i svaka ulazna konekcija ima određenu **težinu**. TLU računa skalarni proizvod ulaznog vektora i odgovarajućih težina ($z = x^T w = x_1 w_1 + x_2 w_2 + ... + x_n w_n$). Nad realnom vrednošću dobijenom iz skalarnog proizvoda primenjuje se *stepena funkcija (engl. step function)* koja daje izlaz iz neurona, $h_w(x) = \text{step}(z)$, gde je $z = x^T w$. Ilustracija ovakvog modela veštačkog neurona je data na slici 4.1.



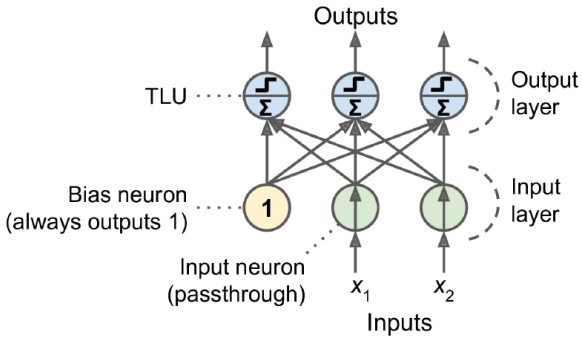
Slika 4.1: Model veštačkog neurona kod perceptrona - TLU

Najčešće korišćena stepena funkcija kod TLU je *Heavyside step function* koja je dana u jednačini (4.1).

$$heavyside(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases} \tag{4.1}$$

Jedan TLU može da se koristi za jednostavnu binarnu klasifikaciju. Računa linearnu kombinaciju elemenata na ulazu, i ako je rezultat veći od zadate granice, na izlazu je 1, tj smatra se da instanca pripada pozitivnoj klasi. U suprotonom se radi o negativnoj klasi (slično kao logistička regresija, na čiju ćemo se primenu kasnije osvrnuti). Trening TLU se svodi na to da se nađu optimalne vrednosti za ulazne težine vektora w, tako da se tačne predikcije daju za što više instanci.

Perceptron se sastoji od jednog sloja TLU-ova, i svaki TLU je povezan sa svakim ulazom. Kada su svi neuroni jednog sloja povezani sa svim neuronima prethodnog sloja, takav sloj se naziva *potpuno povezan sloj ili gusti sloj*. Ulazi Perceptrona se propuštaju kroz posebne propusne neurone, koji se nazivaju *ulazni neuroni*. Oni na izlazu daju šta god im se nalazi na ulazu. Svi ulazni neuroni formiraju *ulazni sloj*. Svim ulaznim neuronima se dodaje još jedan neuron koji se naziva *bias* neuron koji ima konstantan izlaz 1. Perceptron koji ima dva ulaza i tri izlaza dat je na slici 4.2. Ovakav perceptron može da klasifikuje instance simultano u tri različite klase, što ga čini višeizlaznim klasifikatorom.



Slika 4.2: Arhitektura Perceptrona sa dva ulazna neurona, jednim bias neuronom i tri izlazna neurona

Jednačinom (4.2) je moguće efikasno izračunati izlaze sloja veštačkih neurona za nekoliko instanci istovremeno. U jednačini X predstavlja matricu ulaznih instanci, W je matrica težina, koja sadrži težine svih konekcija sem konekcija koje dolaze od bias neurona, b je bias vektor koji sadrži sve težine konekcija između bias neurona i svih neurona sa kojima je povezan, funkcija ϕ se naziva *aktivacionom funkcijom*: ona određuje kada neuron aktivira izlaz. Kod TLU aktivaciona funkcija je stepena funkcija.

$$h_{\mathrm{W,\,b}} = \phi(XW + b) \tag{4.2}$$

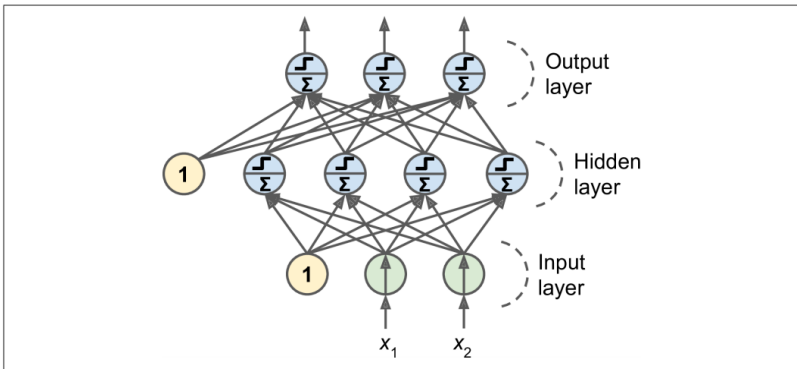
Pravilo za treniranje perceptrona osnažuje konekcije koje smanjuju grešku pri predikciji. Perceptron dobija jednu po jednu trening instancu, i za svaku od njih pravi predikciju. Nakon predikcije, za svaki izlazni neuron koji je napravio pogrešnu predikciju, njemu se osnažuju/povećavaju težine onih konekcija sa ulaza koje bi doprinele pravilnoj predikciji. Pravilo je dato u jednačini (4.3).

$$w_{\mathrm{i,\,j}}{}^{\text{sledeći korak}} = w_{\mathrm{i,\,j}} + \eta(y_{\mathrm{j}} - \hat{y}_{\mathrm{j}})x_{\mathrm{i}} \tag{4.3}$$

U ovoj jednačini, $w_{i,j}$ je težina konekcije između i-tog ulaznog neurona i j-tog izlaznog neurona, x_i je i-ta ulazna vrednost trenutne trening instance, $\hat{y}_j$ je izlaz j-tog izlaznog neurona za trenutnu trening instancu, y_j je prava izlazna vrednost j-tog neurona tj vrednost koju bi trebalo da ima izlazni neuron za trenutnu trening instancu, η je learning rate parametar.

Višeslojni Perceptroni i bekpropagacija

Višeslojni perceptroni se sastoje od jednog sloja propusnih neurona, i jednog ili više slojeva TLU, koji se nazivaju „skriveni slojevi" i poslednjeg izlaznog sloja. Slojevi koji su bliži ulaznom sloju nazivaju se niži ili donji slojevi, a slojevi koji su bliži izlaznom sloju nazivaju se viši slojevi. Svaki sloj sem ulaznog sloja ima svoj bias neuron i svi susedni slojevi su potpuno povezani. Na slici 4.3 je dat prikaz jednog višeslojnog perceptrona.



Slika 4.3: Arhitektura višeslojnog perceptrona

Kada neuronska mreža sadrži više skrivenih slojeva obično se naziva dubokom neuronskom mrežom. U početku razvoja ovog tipa mreža dugo vremena nije bilo moguće naći rešenje za njihovo treniranje. Međutim, 1986. godine grupa naučnika je dala rada kojim su osmislili algoritam bekpropagacije koji se i danas koristi kod treniranja mreža. Srž ovog algoritma je algoritam opadajućeg gradijenta o kojem je ranije bilo reči. U samo dva prolaza kroz mrežu (jedan napred, i jedan unazad) algoritam bekpropagacije može da izračuna gradijent funkcije gubitaka za svaki parametar modela. Jednostavnije rečeno, može da izračuna kako težina svake konekcije i svaki bias neuron treba da budu optimizovani da bi se smanjila greška u predikciji. Kada mreža izračuna ove parametre, postupa kao u slučaju klasičnog algoritma opadajućeg gradijenta, tj „kreće se niz strminu" sve dok ne konvergira ka nekom rešenju. U prolazu unapred model vrši predikciju, i meri grešku u predikciji na kraju. U povratku kroz mrežu meri koliko je svaka od konekcija doprinela grešci i na kraju optimizuje parametre da bi se greška smanjila.

Da bi algoritam funkcionisao bez problema potrebno je promeniti i aktivacionu funkciju neurona. Umesto stepene funkcije, potrebno je koristiti sigmoidnu funkciju. Ova izmena je neophodna jer je stepena funkcija „ravna", nema prevojnih tačaka, nema lokalnih i globalnih minimuma ka kojima bi se moglo kretati. Logistička funkcija ima svuda definisani nenulti izvod i zbog toga je moguće primeniti algoritam opadajućeg gradijenta. Logistička funkcija se i u neuronskim mrežama može koristiti za binarnu klasifikaciju tj određivanje da li instanca pripada nekoj klasi ili ne. Ako je potrebno da raditi višeklasnu klasifikaciju moguće je koristiti kao aktivacionu funkciju,

softmaks funkciju. U narednom delu biće data implementacija neuronskih mreža za binarnu i višeklasnu klasifikaciju korišćenjem TensorFlow biblioteke otvorenog koda. Pored klasifikacija, neuronske mreže se mogu koristiti i za regresiju, kao i za klasifikaciju slika i obradu prirodnih jezika.

4.2 Implementacija veštačkih neuronskih mreža za klasifikaciju

Veštačke neuronske mreže se mogu implementirati u više različitih frejmvorka. Najčešći izbor su biblioteke poput TensorFlow ili PyTorch. TensorFlow je biblioteka otvorenog koga koja sadrži specijalizovane funkcije za kreiranje, treniranje i testiranje neuronskih mreža. Razvijena je u okviru Google Brain projekta, za interne potrebe, a zatim je 2015. godine predstavljena javnosti kao biblioteka otvorenog koda. Danas je uz Pytorch jedna od najpopularnijih biblioteka za kreiranje i treniranje neuronskih mreža. U nastavku će biti data implementacija veštačkih neuronskih mreža TensorFlow bibliotekom.

Primena softmaks funkcije u neuronskim mrežama

Prvi primer mreže će se koristiti za višeklasnu klasifikaciju i koristiće softmaks kao aktivacionu funkciju poslednjeg sloja. Za višeklasnu klasifikaciju biće korišćen već prečišćeni skup podataka *Fashion MNIST*. Ovaj skup podataka se najčešće koristi kada se testira neki novi model, i standard je meru performansi modela. Sadrži preko 70000 slika odeće, svaka slika je veličine 28x28 piksela. Svaka slika ima labelu od 1 do 9 kom tipu odeće pripada. Na slici 4.4 je dat kod za učitavanje skupa podataka.

```
# load dataset and load data
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

Slika 4.4: Kod za učitavanje Fashion MNIST skupa podataka

Zatim je potrebno podeliti skup podataka na trening i test skup i normalizovati podatke. Normalizacija podataka se radi sa vrednošću 255, koliko je maksimalni intenzitet piksela u osmobitnoj grafici. Na slici 4.5 je dat deo koda za podelu skupa na trening i test skup.

```
# scaling the features
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
y_valid, y_train = y_train_full[:5000] / 255.0, y_train_full[5000:] / 255.0
```

Slika 4.5: Podela skupa podataka na trening i test skup

Nakon podele skupa podataka, potrebno je izgraditi model neuronske mreže. U TensorFlow biblioteci se to radi preko Sequential klase koja omogućava da se metodom add dodaju slojevi mreže jedan na drugi. Za svaki sloj se definiše broj neurona u tom sloju, i koju aktivacionu funkciju sloj koristi. Očigledno je da poslednji sloj koristi softmaks funkciju kojom se dobijaju verovatnoće da ta instanca pripada nekoj od klasa.

```
# build the neural network
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation=keras.activations.relu))
model.add(keras.layers.Dense(100, activation=keras.activations.relu))
model.add(keras.layers.Dense(10, activation=keras.activations.softmax))
```

Slika 4.6: Kod za učitavanje Fashion MNIST skupa podataka

Nakon izgradnje modela, potrebno je taj model istrenirati nad skupom podataka. Najpre je potrebno pripremiti model što se radi funkcijom compile(). U funkciji compile je potrebno navesti koja se metoda optimizacije koristi i koja funkcija gubitaka se koristi. Treniranje modela se vrši funkcijom fit() koja ima sličan potpis kao funkcije fit biblioteke scikit-learn. Na slici 4.11 je dat kod za treniranje neuronske mreže.

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=[keras.metrics.sparse_categorical_accuracy]
              )

history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))
```

Slika 4.7: Treniranje neuronske mreže na Fashion MNIST skupom podataka

Posle treniranja modela moguće je koristiti model za predikciju. Najjednostavnije je uzeti nekoliko instanci iz test skupa i videti kako model vrši predikciju za njih. Primer koda za testiranje skupa dat je na slici 4.8.

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
print(y_proba.round(2))
print(y_test[:3])
```

Slika 4.8: Testiranje neuronske mreže nad Fashion MNIST skupom podataka

Primena logističke funkcije u neuronskim mrežama

Za testiranje funkcije logističke funkcije kao aktivacione funkcije kod neuronskih mreža iskoristićemo isti skup podataka, s jednom malom razlikom. Mreža mora biti trenirana za binarnu klasifikaciju i nasumično je izabrano da mreža prepoznaje sve slike sandala (označene labelom 5). U nastavku biće dat samo kod koji se razlikuje u odnosu na implementaciju softmaks funkcije.

Na slici 4.9 je dat deo koda koji deli početni skup podatka na trening i test skup. Zatim se labele za trening i test skupove filtriraju, tako da samo govore da li instanca pripada klasi 5 ili ne.

Nakon podele skupa podataka, potrebno je izgraditi model neuronske mreže. U TensorFlow biblioteci se to radi preko Sequential klase koja omogućava da se metodom add dodaju slojevi mreže jedan na drugi. Za svaki sloj se definiše broj neurona u tom sloju, i koju aktivacionu funkciju sloj koristi. Očigledno je da poslednji sloj koristi logističku funkciju kojom se dobija vrednost 1 ili 0 u zavisnosti da li ta instanca pripada klasi 5.

```
# scaling the features
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
y_valid, y_train = y_train_full[:5000] / 255.0, y_train_full[5000:] / 255.0

y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)
```

Slika 4.9: Podela skupa podataka na trening i test skup

```
# build the neural network
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation=keras.activations.relu))
model.add(keras.layers.Dense(100, activation=keras.activations.relu))
model.add(keras.layers.Dense(1, activation=keras.activations.sigmoid))
```

Slika 4.10: Arhitektura neuronske mreže koja koristi logističku regresiju kao funkciju izlaznog sloja

Nakon izgradnje modela, potrebno je taj model istrenirati nad skupom podataka. Najpre je potrebno pripremiti model što se radi funkcijom compile(). U funkciji compile je potrebno navesti koja se metoda optimizacije koristi i koja funkcija gubitaka se koristi. Treniranje modela se vrši funkcijom fit() koja ima sličan potpis kao funkcije fit biblioteke scikit-learn. Na slici 4.11 je dat kod za treniranje neuronske mreže. Funkcija compile je izostavljena jer je ista kao u prethodom primeru.

```
history = model.fit(X_train, y_train_5, epochs=30, validation_data=(X_valid, y_valid_5))
```

Slika 4.11: Treniranje neuronske mreže na Fashion MNIST skupom podataka

Posle treniranja modela moguće je koristiti model za predikciju. Najjednostavnije je uzeti nekoliko instanci iz test skupa i videti kako model vrši predikciju za njih. Primer koda za testiranje skupa dat je na slici 4.12.

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
print(y_proba.round(2))
print(y_test[:3])
```

Slika 4.12: Testiranje neuronske mreže Fashion MNIST skupom podataka

Glava 5

Zaključak