Usporedba klasičnih algoritama strojnog učenja i neuronske mreže u klasifikacijskim problemima

Sečan, Krešimir

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Usporedba klasičnih algoritama strojnog učenja i neuronske mreže u klasifikacijskim problemima

Završni rad

Mentor:

izv. prof. dr. sc.
Domagoj Matijević

Student:

Krešimir Sečan

Osijek, 2024.

Sadržaj

1 | Uvod

Neuronske mreže su matematički modeli koji su napravljeni po uzoru na ljudski neuronski sustav te kao takve pokušavaju replicirati rad neurona tj. primiti, obradit i proslijediti podatke. Neuronsku mrežu možemo promatrati kao usmjereni aciklički graf, vrhove grafa nazivamo neuronima te je svakom vrhu pridružena aktivacijska funkcija.

Neuronske mreže imaju svoje korijene u 1940-ima, počevši s matematičkim modelom neuronskih mreža koji su izradili Warren McCulloch i Walter Pitts. U 1950-ima i 60-ima doživjeli su daljnji razvoj izumom perceptrona, rane vrste neuronske mreže. Zanimanje je oslabilo tijekom 1970-ih, ali se ponovno oporavilo 1980-ih s algoritmom povratne propagacije, koji je poboljšao učenje.

Tijekom 1980-ih također su se uveli i algoritmi klasifikacije poput metode potpornih vektora i stabla odlučivanja, koji su ponudili fleksibilnija rješenja i postavili temelje za daljnji napredak u strojnom učenju U 21. stoljeću došlo je do brzih napredaka u području strojnog učenja, potaknutih povećanom računalnom snagom i velikim skupovima podataka.

Kao jedan od naprednijih alata današnjice, neuronske mreže imaju široku primjenu poput klasifikacije, regresije, generiranja sadržaja i slično. Iako imaju široku primjenu, ovaj će se završni rad posvetiti problemima klasifikacije. Kako bismo istaknuli prednosti neuronskih mreža, u odnosu na tradicionalne metode klasifikacije, usporedit ćemo rad neuronske mreže s već poznatim algoritmima klasifikacije.

Ovaj rad sastoji se od tri poglavlja. U prvom poglavlju reći ćemo više o zadacima klasifikacije i klasičnim algoritmima koji se primjenjuju za takve zadatke. U drugom poglavlju posvetit ćemo se neuronskim mrežama, objasnit ćemo osnovne pojmove i principe rada te neke osnovne tehnike poboljšanja. Posljednje poglavlje uključuje razvoj tri različita modela, nakon čega slijedi njihovo treniranje, evaluacija i usporedba.

2 | Zadaci klasifikacije

Zadaci klasifikacije su zapravo problemi koji se svode na podjelu ulaznih podataka na dvije ili više kategorija. Cilj je da model uči iz ovih podataka i napravi točna predviđanja kada mu se predstave nove, neviđene instance. Kako bi model bio što učinkovitiji ključno je izabrati što bolji algoritam.

2.1 Linearni klasifikator

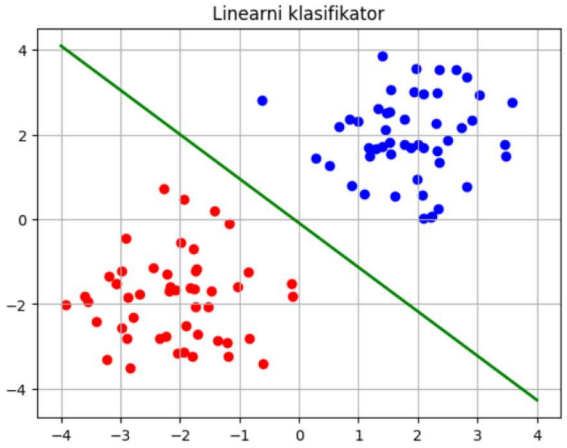
Kako bi smo pobliže objasnili i opisali zadatke klasifikacije pogledat ćemo linearni klasifikator za izlazne varijable $y \in \{-1, 1\}$.

Zadana nam je funkcija model $h_\Theta = \text{sign}(\Theta^T \cdot x)$, pri čemu su parametri

$$\Theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1},$$

a funkcija sign definirana kao:

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$



Slika 2.1: Prikaz linearnog klasifikatora

2.2 Logistička regresija

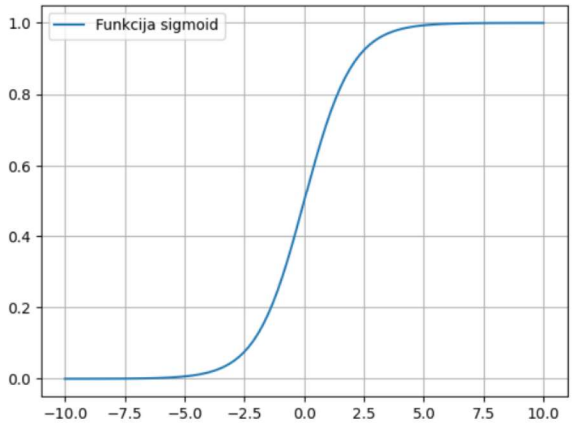
U logističkoj regresiji [3, p. 126,127] promatrat ćemo skup funkcija

$$h_\Theta : R^n \to [0,1].$$

Funkcija model definirana je kao $h_\Theta(x) = \sigma(\Theta^T \cdot x)$,gdje je

$$\Theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix},$$

pri čemu je $\sigma : R \to R, \sigma(z) = \frac{1}{1+e^{-z}}$ i naziva se sigmoid ili logistička funkcija. Ime sigmoid proizlazi iz toga što je graf same funkcije u obliku slova S (2.2).



Slika 2.2: Funkcija sigmoid

Model logističke regresije koristi funkciju sigmoid kako bi pretvorio $\Theta^T \cdot x$ u vjerojatnost, što omogućava donošenje odluke o klasifikaciji na temelju predefiniranog praga. Dakle, Funkciju $h_\Theta(x)$ možemo promatrati kao $h_\Theta(x) = P(y = 1|x; \Theta)$.

2.2.1 Funkcija maksimalne vjerodostojnosti

Kako bismo maksimizirali vjerojatnost da se svaki podatak klasificira u odgovarajuću klasu koristit ćemo funkciju maksimalne vjerodostojnosti (eng. Maximum likelihood function). Želimo maksimizirati funkciju:

$$J'(\Theta) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)}; \Theta),$$

pri čemu je $y \in \{-1, 1\}$. Ekvivalentno je maksimizirati funkciju $\log(J'(\Theta))$ (eng. loglikelihood function) jer je logaritamska funkcija monotono rastuća, odnosno minimizirati $-\log(J'(\Theta))$.

$$\begin{aligned} J(\Theta) &= -\log(J'(\Theta)) \\ &= \sum_{i=1}^{m} -\log(P(y^{(i)}|x^{(i)}; \Theta)) \\ &= \sum_{i=1}^{m} -\log(\sigma(y^{(i)}\Theta^T x^{(i)})). \end{aligned}$$

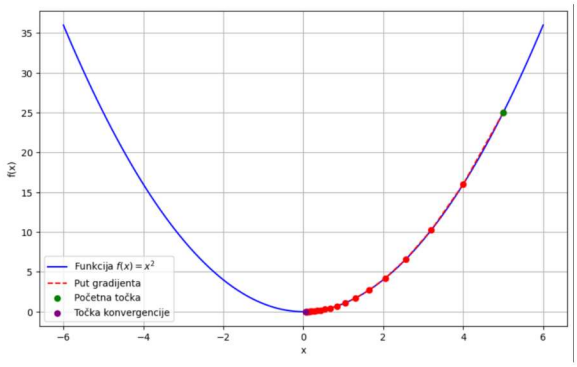
Budući da funkcija sigmoid zadovoljava svojstvo $\sigma(-a) = 1 - \sigma(a)$ [2, p. 197] slijedi:

$$J(\Theta) = \sum_{i=1}^{m} \log(1 + e^{-y^{(i)}\Theta^T x^{(i)}}).$$

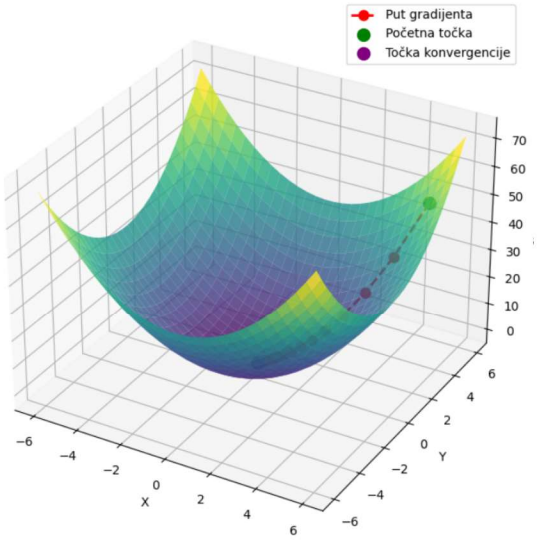
Budući da je $J(\Theta)$ koveksna funkcija za minimizaciju možemo koristiti gradijentnu metodu.

Algoritam 1 Gradijentna metoda

1: **procedure** GRADIENT
2: $\Theta \leftarrow$ početna vrijednost
3: **while** uvijet konvergencije **do**
4: $\Theta_j \leftarrow \Theta_j - \alpha \frac{d}{d\Theta_j} J(\Theta), \forall j$



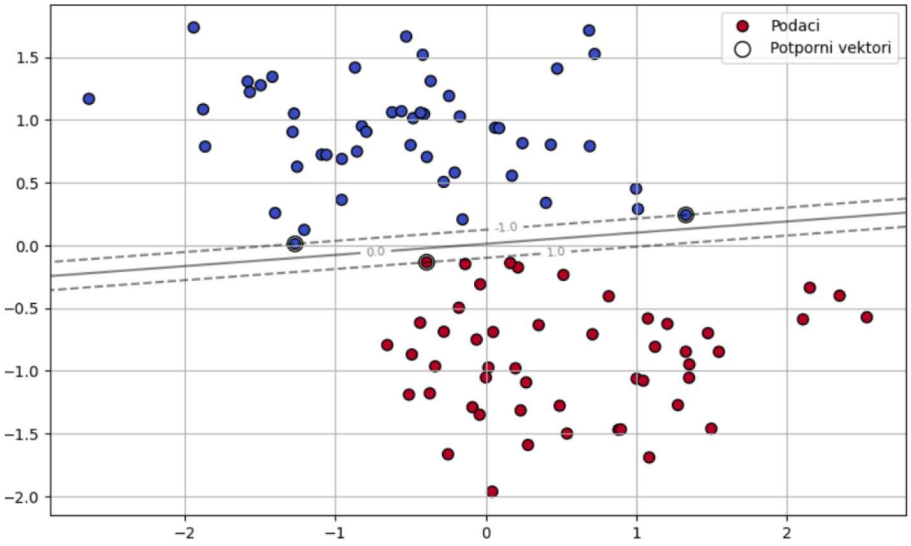
Slika 2.3: Prikaz gradijentne metode u dvije dimenzije



Slika 2.4: Prikaz gradijentne metode u tri dimenzije

2.3 Metoda potpornih vektora

Metoda potpornih vektora (eng. SVM, Support vector machine) koristi drugačiji pristup za klasifikaciju podataka od logističke regresije. Cilj je pronaći najkraću udaljenost između podataka iz skupa za treniranje i hiperravnine, takozvanu geometrijsku marginu γ^*. Hiperravninu ćemo definirati kao linearnu kombinaciju potpornih vektora, koji su zapravo podatkovne točke koje leže najbliže hiperravnini 2.5. Dakle, jednadžba hiperravnine može se predstaviti kao zbroj potpornih vektora s različitim težinama, pri čemu metoda potpornih vektora (eng. SVM) određuje te težine kako bi se maksimizirala geometrijska margina.



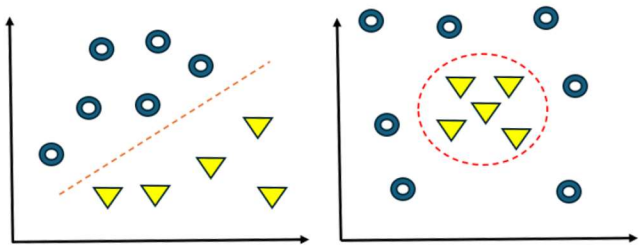
Slika 2.5: Prikaz potpornih vektora

Pretpostavit ćemo da su podaci linearno separabilni i dolaze iz $y \in \{-1, 1\}$. Podaci $\{(x^{(i)}, y^{(i)}), i = 1, .., m\}$ su linearno separabilni ukoliko postoji

$$\overline{\Theta} = \begin{bmatrix} \overline{\theta_0} \\ \overline{\theta_1} \\ \vdots \\ \overline{\theta_n} \end{bmatrix} \in \mathbb{R}^{n+1},$$

takav da vrijedi:

$$y^{(i)} \cdot \left(\overline{\Theta}^T \cdot \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix} \right) > 0, \quad \forall i = 1, \ldots, m.$$



Slika 2.6: Linearno separabilni i ne linearno separabilni podaci

Neka je model funkcija:

$$h_\Theta(x) = \text{sign}(\Theta^T \cdot x).$$

Maksimizirat ćemo γ^*, uz uvjet $y^{(i)}\Theta^T x^{(i)} \geq \gamma, \forall i = 1, .., m$. Ekvivaletno je promatrati problem minimizacije $\frac{1}{\gamma^*} = \frac{||\overline{\Theta}||}{\gamma}$, uz isti uvjet.

Ukoliko minimiziramo $(\frac{||\overline{\Theta}||}{\gamma})^2$ minimum se ne mijenja te se problem svodi na

$$\min(\frac{1}{2} \cdot \frac{||\overline{\Theta}||^2}{\gamma^2}), \quad y^{(i)}\Theta^T x^{(i)} \geq \gamma, \forall i = 1, .., m,$$

$$\min(\frac{1}{2} \cdot ||\frac{\overline{\Theta}}{\gamma}||^2), \quad y^{(i)}(\frac{\Theta}{\gamma})^T x^{(i)} \geq 1, \forall i = 1, .., m.$$

Skaliranje parametra γ ne utječe na hiperravninu, stoga ga možemo fiksirati na 1 i optimizirati Θ. Minimiziramo kvadratičnu konveksnu funkciju izrazom:

$$\min(\frac{1}{2} \cdot ||\overline{\Theta}||^2), \quad y^{(i)}\Theta^T x^{(i)} \geq 1, \forall i = 1, .., m.$$

Ako podaci nisu linearno separabilni uvodimo slack varijable $\xi^{(i)}, i = 1, ..., m$. Ukoliko je slack varijabla $\xi^{(i)} > 0$, i-ti podatak se nalazi s krive strane geometrijske margine. Uvedimo u formulaciju slack varijable:

$$\min \frac{1}{2} \cdot ||\overline{\Theta}||^2 + C \cdot \sum_{i=1}^{m} \xi^{(i)} \qquad y^{(i)}\Theta^T x^{(i)} \geq 1 - \xi^{(i)}, \quad \xi^{(i)} \geq 0 \tag{2.1}$$

Geometrijsku marginu računamo kao $\gamma^* = \frac{1}{||\overline{\Theta^*}||}$, gdje je (Θ^*, ξ^*) optimalno rješenje jednadžbe (2.1).

2.4 Klasifikacija u više klasa

Binarna klasifikacija široke je primjene u praksi, ali često je potrebno podatke podijeliti u više od dvije kategorije. Dodatak još jedne kategorije, iako se čini kao mala promjena u praksi, zapravo ima dosta veliki utjecaj na algoritme i njihovo izvođenje. Ulazni podaci poprimaju novi oblik

$$(x^{(i)}, y^{(i)}), x^{(i)} \in R^n, y^{(i)} \in \{1, 2, ..., k\}.$$

Generalizirat ćemo koncept logističke regresije u takozvanu softmax regresiju. Softmax funkciju definiramo kao $\rho : R^k \to R^k$,

$$\rho(\vec{z}) = \begin{bmatrix} \rho(\vec{z})_1 \\ \vdots \\ \rho(\vec{z})_k \end{bmatrix} = \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_k \end{bmatrix} \quad t.d. \quad \rho_j = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}} \ , \ \text{uočimo} \ \sum_{i=1}^{k} \rho_i = 1$$

Definiramo model funkciju $h_\Theta : R^n \to R^k$,

$$h_\Theta(x) = \rho \begin{bmatrix} \Theta^{(1)T} \cdot x \\ \vdots \\ \Theta^{(k)T} \cdot x \end{bmatrix} = \begin{bmatrix} \rho(\Theta^{(1)T} \cdot x)_1 \\ \vdots \\ \rho(\Theta^{(k)T} \cdot x)_k \end{bmatrix} = \sum_{j=1}^{k} e^{\Theta_j T} x \begin{bmatrix} e^{\Theta^{(1)T} \cdot x} \\ \vdots \\ e^{\Theta^{(k)T} \cdot x} \end{bmatrix} = \begin{bmatrix} P(y=1|x;\Theta) \\ \vdots \\ P(y=k|x;\Theta) \end{bmatrix}.$$

Dakle, funkcija model bit će oblika $h_\Theta : R^n \to R^k$ pri čemu vrijedi:

$$h_\Theta(x) = \begin{bmatrix} P(y=1|x;\Theta) \\ \vdots \\ P(y=k|x;\Theta) \end{bmatrix} \text{ , } i \text{ vrijedi } \sum_{i=1}^{k} P(y=i|x;\Theta) = 1.$$

Θ je matrica dimenzija $(n+1) \times k$, dana s

$$\Theta = \begin{bmatrix} \vdots & & \vdots \\ \Theta^{(1)} & \cdots & \Theta^{(k)} \\ \vdots & & \vdots \end{bmatrix},$$

kako bismo odredili što optimalniji parametar Θ maksimizirat ćemo funkciju

$$J'(\Theta) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)};\Theta),$$

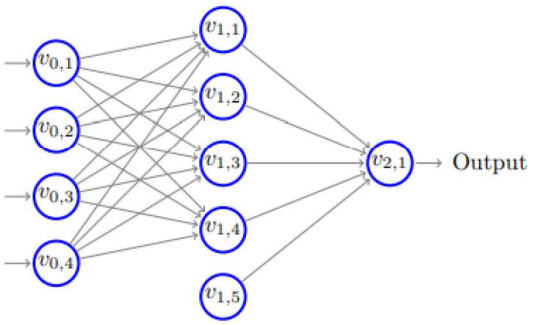
potreban nam je gradijent:

$$\nabla J(\Theta) = \begin{bmatrix} \vdots & & \vdots \\ \nabla\Theta^{(1)} J(\Theta) & \cdots & \nabla\Theta^{(k)} J(\Theta) \\ \vdots & & \vdots \end{bmatrix}, \quad \nabla\Theta^{(j)} J(\Theta) = \begin{bmatrix} \frac{\partial J(\Theta)}{\partial \Theta_1^{(j)}} \\ \vdots \\ \frac{\partial J(\Theta)}{\partial \Theta_k^{(j)}} \end{bmatrix}.$$

Koristit ćemo gradijentnu metodu (1), pri čemu ćemo unutar petlje imati: $\Theta \leftarrow \Theta - \alpha \nabla J(\Theta)$

3 | Neuronske mreže

Neuronsku mrežu možemo definirati kao usmjereni aciklički graf G =(V,E) s funkcijom težine $\Theta : E \to R$. Vrhove grafa (V) nazivamo neuronima, a svakom neuronu je pridružena aktivacijska funkcija $a : R \to R$. Odabirom vrhova i bridova (E) zapravo određujemo arhitekturu neuronske mreže. Posvetiti ćemo se neuronskim mrežama s propagacijom unaprijed (eng. feed-forward neural network) te detaljno objasniti njihove komponente, arhitekturu i principe rada.



Slika 3.1: Prikaz neuronske mreže, [3, p. 270]

3.1 Funkcije gubitka

Funkcijom gubitka (eng. loss function) zapravo mjerimo koliko su blizu dobivene vrijednosti ciljanim vrijednostima. Cilj samog treniranja neuronske mreže je minimizirati funkciju gubitka. Budući da se fokusiramo na probleme klasifikacije, definirat ćemo funkciju višeklasne unakrsne entropije (eng. cross entropy loss). Ova se funkcija koristi za izračun pogreške između predviđene vjerojatnosti raspodjele klasa i ciljanih oznaka enkodiranih u one-hot obliku.

$$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} y_{ij}\log(\overline{y_{ij}} + \epsilon),$$

pri čemu je $\overline{y_{ij}}$ predviđena vjerojatnost da podatak i pripada klasi j, a y_{ij} one-hot enkodirana oznaka za podatak i i klasu j.

3.2 Slojevi

Neuronska mreža sastoji se od minimalno dva sloja ulaznog i izlaznog, svi slojevi između ulaznog i izlaznog nazivaju se skriveni slojevi (eng. hidden layers). Možemo izdvojiti neke specifične slojeve poput potpuno povezanog sloja (eng. fully connected layer or dense layer). Ovaj se sloj ističe jer je svaki neuron povezan sa svim neuronima iz prethodnog sloja. Također postoji i sloj za normalizaciju (eng. normalization layer) koji skalira podatke što omogućuje brže i stabilnije treniranje modela. Svaki neuron uzima podatke od prethodnog sloja, primjenjuje funkciju aktivacije i prosljeđuje podatke idućem sloju.

3.3 Aktivacijske funkcije

Bez obzira na broj i vrstu slojeva, bez aktivacijske funkcije naš bi model bio sposoban samo modelirati linearno preslikavanje. Neke poznatije funkcije aktivacije:

- Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hiperbolna tangens funkcija $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- ReLu(Rectified Linear Units) $R(x) = \max\{0, x\}$

Aktivacijske funkcije primjenjuju transformaciju na zbroj ulaza i proizvode izlaz koji se zatim prosljeđuje sljedećem sloju. Oznakom $a_i^{(j)}$ označavamo aktivacijsku funkciju i-tog neurona u j-tom sloju mreže, u matričnom zapisu

$$a^{(j)} = \begin{bmatrix} a_0^{(j)} \\ \vdots \\ a_{s_j}^{(j)} \end{bmatrix}.$$

Ako neuronska mreža ima l slojeva, a svaki sloj ima s_j neurona $j = 1, ..., l$, tada je aktivacijska funkcija za j-ti sloj dana jednadžbom $a^{(j)} = a(\Theta^{(j-1)} \cdot a^{(j-1)}), j = 2, 3, ..., l$. Možemo vidjeti da kako bismo izračunali i+1 sloj moramo prvo izračunati i-ti sloj, taj proces nazivamo propagacijom unaprijed (eng. forward propagation).

3.4 Algoritam povratne propagacije

Algoritam povratne propagacije (eng. backpropagation algorithm) [2, p. 241] ključan je za učenje neuronske mreže, a provodi se kako bi se poboljšalo treniranje neuronske mreže i ostvarili bolje rezultate pri testiranju. Nakon provođenja propagacije unaprijed, provodimo algoritam povratne propagacije. Algoritam računa pogrešku, točnije razliku između dobivene vrijednosti pri izlazu i ciljane vrijednosti. Nakon toga pogreška se prenosi natrag kroz mrežu i prilagođavaju

se težine kako bi se pogreška minimizirala. Proces prilagodbe težina provodi se često pomoću algoritama za optimizaciju.

Radi lakšeg raspisa pretpostavit ćemo da naša mreža u svakom sloju ima samo jedan neuron. Označimo s $a^{(L)}$ aktivacijsku funkciju za L-ti sloj te s y ciljanu vrijednost. Funkciju troška za jedan podatak označimo s

$$C_0 = (a^{(L)} - y)^2.$$

Označimo sa

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)},$$

pri čemu je $w^{(L)}$ težina, a $b^{(L)}$ bias.

Definiramo aktivacijsku funkciju jednadžbom $a^{(L)} = \sigma(z^{(L)})$, gdje je σ sigmoid funkcija. Zanima nas koliko određeni parametri utječu na funkciju troška C_0, točnije zanima nas

$$\frac{\partial C_0}{\partial w^{(L)}}, \quad \frac{\partial C_0}{\partial b^{(L)}}, \quad \frac{\partial C_0}{\partial a^{(L-1)}}.$$

Budući da vidimo da je funkcija C_0 povezana s aktivacijskom funkcijom $a^{(L)}$, a aktivacijski funkcija je povezana s funkcijom $z^{(L)}$ koja je povezana sa funkcijom težine $w^{(L)}$. Pravilom lanca(eng. chain rule) traženu derivaciju možemo svesti na

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}},$$

pri čemu su derivacije:

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y),$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}),$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}.$$

Ovdje smo prikazali funkciju troška C_0 za samo jedan podatak, kako bismo dobili pravu funkciju troška moramo uzeti prosjek funkcija troška za svih n podatka.

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}.$$

Isti postupak možemo primijeniti kako bismo odredili $\frac{\partial C_0}{\partial b^{(L)}}$,

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}},$$

pri čemu je

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1.$$

Pogledajmo još kako aktivacijska funkcija iz prethodnog sloja utječe na funkciju troška.

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}},$$

pri čemu je

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)}.$$

Ovaj postupak možemo ponavljati kako bismo vidjeli kako parametri iz prethodnih slojeva utječu na funkciju troška. Isto tako računamo derivacije koje nam određuju gradient koji nam pomaže pri minimizaciji funkcije troška.

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}.$$

Poopćimo naš raspis na neuronsku mrežu u kojoj se L-ti sloj sastoji od m neurona, a L-1 sloj od k neurona. Pretpostavimo da je $y = y_0, ..., y_{m-1}$. Tada je

$$C_0 = \sum_{j=0}^{m-1} (a_j^{(L)} - y_j)^2,$$

$$z_j^{(L)} = \sum_{i=0}^{m-1} w_{ji}^{(L)} a_i^{(L-1)} + b_j^{(L)},$$

$$a_j^{(L)} = \sigma(z_j^{(L)}).$$

Derivacija poprima oblik:

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}.$$

Glavna promjena u odnosu na neuronsku mrežu s jednim neuronom po sloju, događa se pri računanju derivacije ovisne o jednoj aktivacijskoj funkciji iz prethodnog sloja. To se događa zbog toga što neuron s danom aktivacijskom funkcijom utječe na funkciju troška kroz više puteva pa derivacija izgleda ovako:

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{k-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}.$$

Komponente gradijenta su oblika:

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}.$$

3.5 Optimizacijski algoritmi

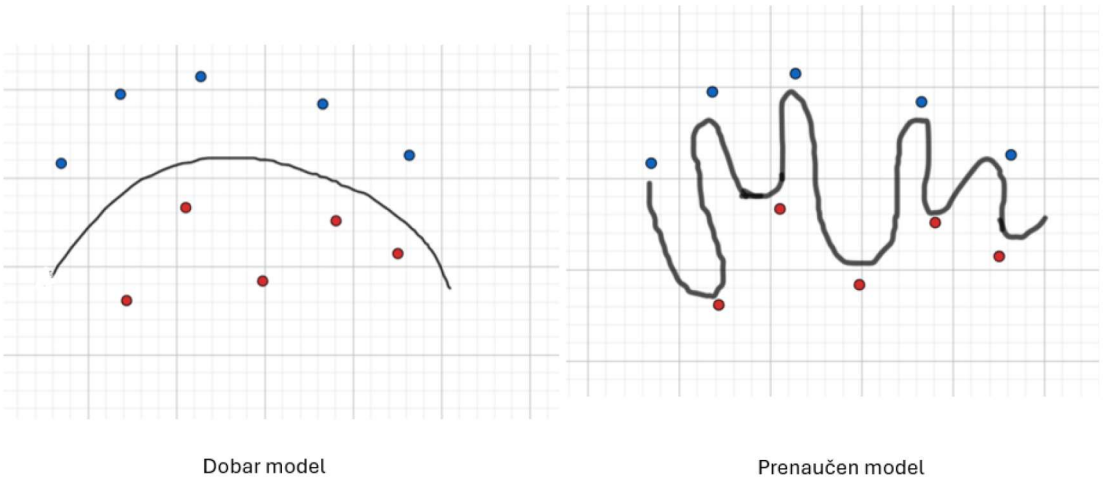
Optimizacijskim algoritmima želimo ubrzati pronalaženje optimalnih težina kako bismo minimizirali funkciju gubitka. Jedan od osnovnih algoritama za optimizaciju je već spomenuta gradijentna metoda (1). Iako efektivna, gradijentna metoda može stagnirati u lokalnim minimumima kada je gradijent blizu nule. Napredne tehnike optimizacije poput Adam-a nemaju taj problem. Adam (adaptive moment estimation) [1, p. 140] je popularan izbor optimizacijskog algoritma jer omogućuje brzu konvergenciju i prilagodbu stope učenja za svaki parametar. Adam je nastao kombiniranjem prednosti dvaju algoritma za optimizaciju AdaGrad i RMSProp. Stopa učenja prilagođava se pomoću dva pokretna prosjeka za svaki parametar, to jest srednje vrijednosti gradijenta i srednje vrijednosti kvadrata gradijenta. Težine se ažuriraju prema formuli:

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{v_k} + \epsilon} \cdot s_k,$$

pri čemu je α stopa učenja, s_k srednja vrijednost gradijenta, v_k srednja vrijednost kvadrata gradijenta i w_k težina u koraku k.

3.6 Regularizacija

Sama ideja regularizacije dolazi iz potrebe kontroliranja utjecaja parametra Θ u funkciji h_Θ kako bi riješili problem prenaučenosti (eng. overfitting). Problem prenaučenosti javlja se kada model funkcija sadrži prevelik broj varijabli i pretjerano je prilagođena trening podacima 3.2.



Slika 3.2: problem prenaučenosti

Regularizacija se provodi dodavanjem regularizacijske funkcije $\Phi : R^n \rightarrow R$ u kriterijsku funkciju. U kontekstu klasifikacije funkcija gubitka često se naziva i

kriterijska funkcija. Dakle, primjenom regularizacije kriterijska funkcija poprima oblik $J(\Theta) + \Phi(\Theta)$. Unosom ove male promjene u algoritam omogućena je bolja generalizacija modela te samim time i bolje performanse na testnim podacima.

3.6.1 L_2 regularizacija

Tihonovljeva regularizacija [3, p. 174] ili L_2 regularizacija definirana je kao

$$\Phi(\Theta) = \lambda \cdot ||\Theta||^2,$$

pri čemu je $\lambda \geq 0$ hiperparametar regularizacije. Primjenimo L_2 regularizaciju na softmax regresiju, pri čemu je $y^{(i)} \in \{1, 2, ..., k\}$. Tada je kriterijska funkcija oblika:

$$J(\Theta) = -\sum_{i=1}^{m}\sum_{j=1}^{k} I^j(y^{(i)}) \cdot \log \frac{e^{\Theta^{(j)T}x^{(i)}}}{\sum_{r=1}^{k} e^{\Theta^{(j)T}x^{(i)}}} + \lambda \cdot \sum_{i=1}^{m}\sum_{j=1}^{k} \Theta_{ij}^2, \quad I^j(x) = \begin{cases} 1, x = j \\ 0, x \neq j \end{cases}.$$

3.6.2 Dropout tehnika regularizacije

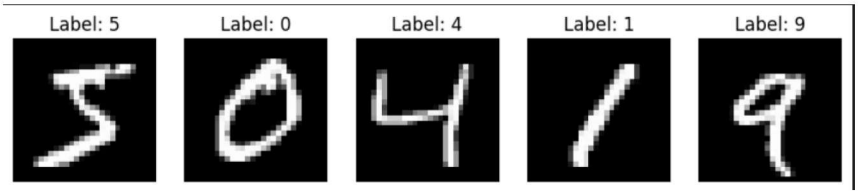
Ova tehnika regularizacije izbacuje dijelove mreže pri procesu treniranja kako bi spriječili prenaučenost modela. Tijekom svakog prolaza naprijed i natrag nasumično se zanemaruju neki neuroni, sprječavajući neuronsku mrežu da postane previše ovisna o određenim neuronima. Hiperparametar vjerojatnost dropout-a (eng. dropout rate) p određuje koliki će postotak neurona biti isključen.

Kako bi mreža imala što bolje rezultate pri testiranju su svi neuroni aktivni. Možemo kompenzirati učinak dropout-a tokom treniranja, tako da skaliramo težine za vjerojatnost da je određeni neuron bio aktivan tokom treninga.

4 | Usporedba neuronske mreže i algoritama klasifikacije

4.1 Podaci

Podaci su temelj strojnog učenja, kvaliteta i kvantiteta podataka direktno utječe na učenje samog modela. Zbog toga podaci trebaju biti što reprezentativniji kako bi se modeli izgradili i usavršili. Sami podaci u konačnici određuju i uspiešnost modela. U procesu usporedbe koristit ćemo skup podata MNIST, koji se sastoji od 70000 znamenaka napisanih rukom. Znamenke dolaze u formatu slika u sivim tonovima veličine 28x28 piksela. Sami pikseli predstavljeni su brojevima od 0 do 255, dok su pripadne oznake u obliku brojeva od 0 do 9.



Slika 4.1: Prikaza nekoliko podataka iz MNIST-a

4.2 Model logističke regresije

Model logističke regresije implementirali smo u pythonu koristeći sklearn biblioteku [4] .

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,  classification_report


mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target
y = y.astype(int)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression( multi_class='multinomial', max_iter
    =1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred, digits=3)


print(class_report)
```

Rezultati koji smo dobili su:

	precision	recall	f1-score	support
0	0.957	0.958	0.957	1343
1	0.946	0.973	0.959	1600
2	0.899	0.893	0.896	1380
3	0.903	0.888	0.895	1433
4	0.922	0.918	0.920	1295
5	0.881	0.875	0.878	1273
6	0.933	0.945	0.939	1396
7	0.921	0.941	0.931	1503
8	0.900	0.860	0.880	1357
9	0.894	0.901	0.898	1420
accuracy			0.916	14000
macro avg	0.916	0.915	0.915	14000
weighted avg	0.916	0.916	0.916	14000

Slika 4.2:

4.3 Model potpornih vektora

Model je implementiran u pythonu, također koristeći sklearn biblioteku [4].

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score

mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target
y = y.astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_linear = SVC(kernel='linear', C=1)
svm_linear.fit(X_train, y_train)
y_pred = svm_linear.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Linear SVM accuracy: {accuracy:.4f}")
print(f"Linear SVM precision: {precision:.4f}")
print(f"Linear SVM recall: {recall:.4f}")
print(f"Linear SVM F1-score: {f1:.4f}")
```

Rezultati koje smo dobili su:

```
Linear SVM accuracy: 0.9211
Linear SVM precision: 0.9214
Linear SVM recall: 0.9211
Linear SVM F1-score: 0.9210
```

Slika 4.3:

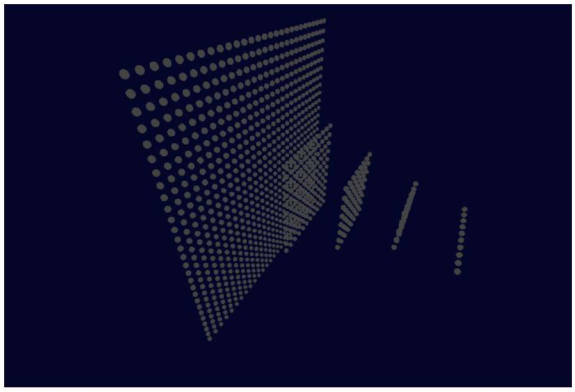
4.4 Neuronska mreža

4.4.1 Model

Neuronska mreža implementirana je u JavaScriptu, korištenjem TensorFlow.js biblioteke. Mreža se sastoji od pet potpuno povezanih slojeva. Ulazni sloj je veličine 28x28, dok je izlazni veličine 10 neurona jer imamo 10 kategorija. Aktivacijska funkcija u svim slojevima, osim predzadnjeg je ReLu dok je u predzadnjem sloju aktivacijska funkcija softmax. Za optimizaciju koristimo stohastički gradijentni spust, a funkcija troška je funkcija unakrsne entropije.

```
function createModel(layerSizes) {
    if (!Array.isArray(layerSizes) || layerSizes.length === 0) {
        throw new Error('layerSizes must be a non-empty array');
    }

    const model = new CustomModel();
    layerSizes.forEach((size, index) => {
```



Slika 4.4: Implementirana neuronska mreža

```
        if (index === 0) {
            model.add(tf.layers.dense({
                units: size,
                activation: 'relu6',
                inputShape: [layerSizes[0]]
            }));
        } else if (index === layerSizes.length - 1) {
            model.add(tf.layers.dense({
                units: size,
                activation: 'softmax'
            }));
        } else {
            model.add(tf.layers.dense({
                units: size,
                activation: 'relu6'
            }));
        }
    });

    model.compile({
        optimizer: 'sgd',
        loss: 'categoricalCrossentropy',
        metrics: ['accuracy']
    });

    return model;
}
```

4.4.2 Treniranje modela

Trenirat ćemo našu neuronsku mrežu na 10 epoha, pri čemu će veličina grupe (eng. Batchsize) biti 32.

```
async function trainModel(trainData, trainLabels, epochs = 10,
    batchSize = 32) {
    if (!trainData || !trainLabels) {
```

```
3           throw new Error('Training data and labels must be provided
    ');
4       }
5
6       let finalAccuracy;
7
8       await model.fit(trainData, trainLabels, {
9           batchSize,
10          epochs,
11          validationSplit: 0.2,
12          callbacks: {
13              onEpochEnd: (epoch, logs) => {
14                  $('#epoch-info').text(`Epoch ${epoch + 1}: loss = $
    {logs.loss.toFixed(4)}, accuracy = ${logs.acc.toFixed(4)}`);
15                  console.log(`Epoch ${epoch + 1}: loss = ${logs.loss
    .toFixed(4)}, accuracy = ${logs.acc.toFixed(4)}`);
16                  finalAccuracy = logs.acc;
17              },
18              onTrainEnd: () => {
19                  $('#epoch-info').hide();
20                  $('#final-score').text(`Training finished. Final
    accuracy: ${finalAccuracy.toFixed(4)}`).show();
21                  $('.spinner-border').hide();
22                  $('#accuracy-display').text(`Accuracy: ${(
    finalAccuracy * 100).toFixed(2)}%`).show();
23              }
24          }
25      });
26  }
```

Prikazat ćemo proces treniranja po epohama, gdje je za svaku epohu prikazana vrijednost funkcije gubitka i preciznost modela na validacijskom setu.

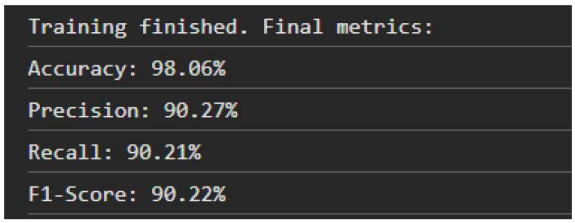
```
Epoch 1: loss = 0.0727, accuracy = 0.4090
Epoch 2: loss = 0.0419, accuracy = 0.7232
Epoch 3: loss = 0.0272, accuracy = 0.8367
Epoch 4: loss = 0.0204, accuracy = 0.8769
Epoch 5: loss = 0.0163, accuracy = 0.9035
Epoch 6: loss = 0.0137, accuracy = 0.9211
Epoch 7: loss = 0.0116, accuracy = 0.9336
Epoch 8: loss = 0.0101, accuracy = 0.9442
Epoch 9: loss = 0.0086, accuracy = 0.9529
Epoch 10: loss = 0.0075, accuracy = 0.9607
```

Slika 4.5: Prikaz procesa treniranja po epohama

4.4.3 Evaluacija

Zbog načina implementacija već tokom faze treniranja vidjeli smo kako parametar točnosti(eng. accuarcy) raste, dok vrijednost funkcije gubitka opada. Nakon tre-

niranja mreže provedena je faza testiranja te je mreža ostvarila sljedeće rezultate.



Slika 4.6: Rezultati nakon testiranja

Funkcija koju smo koristili pri evaluaciji modela:

```
1
2     function calculateMetrics(trueLabels, predictedLabels,
    numClasses) {
3         const trueLabelsArray = trueLabels.arraySync();
4         const predictedLabelsArray = predictedLabels.arraySync();
5
6         const confusionMatrix = Array.from({ length: numClasses },
    () => Array(numClasses).fill(0));
7         for (let i = 0; i < trueLabelsArray.length; i++) {
8             confusionMatrix[trueLabelsArray[i]][
    predictedLabelsArray[i]] += 1;
9         }
10
11        let precisionSum = 0;
12        let recallSum = 0;
13        let f1ScoreSum = 0;
14        let accuracySum=0;
15
16        for (let i = 0; i < numClasses; i++) {
17            const truePositive = confusionMatrix[i][i];
18            const falsePositive = confusionMatrix.reduce((sum, row)
     => sum + row[i], 0) - truePositive;
19            const falseNegative = confusionMatrix[i].reduce((sum,
    value) => sum + value, 0) - truePositive;
20            const trueNegative = trueLabelsArray.length -
    truePositive - falsePositive - falseNegative;
21
22            const accuracy = (truePositive + trueNegative) / (
    truePositive + trueNegative + falsePositive + falseNegative + 1e
    -10);
23            const precision = truePositive / (truePositive +
    falsePositive + 1e-10);
24            const recall = truePositive / (truePositive +
    falseNegative + 1e-10);
25            const f1Score = 2 * (precision * recall) / (precision +
     recall + 1e-10);
26
27            precisionSum += precision;
28            recallSum += recall;
29            f1ScoreSum += f1Score;
30            accuracySum += accuracy;
```

```
31          }
32
33          return {
34              accuracy: accuracySum/ numClasses,
35              precision: precisionSum / numClasses,
36              recall: recallSum / numClasses,
37              f1Score: f1ScoreSum / numClasses
38          };
39      }
```

4.5 Rezultati

Treniranjem i evaluacijom tri različita modela na skupu podataka MNIST postigli smo visoke parametre točnosti. Visoka točnost (eng. accuarcy) ukazuje na to da su modeli ispravno klasificirali veliku većinu instanci, dok veličina ostalih parametara ukazuje nam da je model učinkovit u identificiranju relevantnih slučajeva isto kao i u preciznosti svojih predviđanja.

Model neuronske mreže tokom treniranja na validacijskom setu postigao je točnost od 0.9607, dok je na testnom setu točnost iznosila 0.9886. Model logističke regresije postigao je točnost od 0.916, a model potpornih vektora točnost od 0.9211. Možemo vidjeti kako neuronska mreža ima znatno veću točnost od druga dva modela, ali ima manje ostale parametre.

Model neuronske mreže u parametrima preciznosti (eng. precision), odziva (eng. recall) i F1-score imao je malo slabije rezultate od ostalih modela, što nam sugerira da iako je neuronska mreža općenito točna postoje načini da se poboljša. Navedeni parametri su kod modela neuronske mreže bili oko 0.902, dok su kod modela logističke regresije iznosili 0.916, a kod modela potpornih vektora 0.921.

Najveća razlika među modelima je vrijeme treniranja. Model potpornih vektora imao je najduže vrijeme treniranja 3 minute i 40 sekundi, dok je vrijeme treniranja neuronske mreže i modela logističke regresije iznosilo oko 2 minute. Model logističke regresije, iako je imao brže vrijeme konvergencije od modela potpornih vektora, kako bi konvergirao morao je u pozadini koristiti optimizacijski algoritam LBFGS (eng. Limited-memory Broyden–Fletcher–Goldfarb–Shanno). Ovaj optimizacijski algoritam ubraza vrijeme treninga i smanjuje korištenje memorije, što je pomoglo modelu logističke regresije pri procesu treniranja i ubrzalo konvergenciju.

Svi modeli su se pokazali vrlo dobrima za dani zadatak klasifikacije. Međutim, s obizorm na vrijeme treniranja i parametar točnosti te skalabilnost neuronske mreže na veći skupa podataka i mogućnost dodatnog poboljšanja modela mijenjanjem arhitekture, funkcija aktivacija i slično, neuronska mreža bila bi najpogodnija za dani problem.

Literatura

[1] C. C. Aggarwal, *Neural Networks and Deep Learning (1st ed. 2018)*, Springer.

[2] C. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag, Berlin, 2006.

[3] S. Shalev-Shwartz, S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge Press, 2014.

[4] *Scikit-learn: Machine Learning in Python*, dostupno na `https://scikit-learn.org/stable/`.

Sažetak

U ovom radu razmotrit ćemo zadatke klasifikacije, opisati neke od algoritama koji se koriste za klasifikaciju te opisati neuronsku mrežu. Nakon toga, implementirat ćemo i istrenirati modele za algoritam logističke regresije, metode potpornih vektora i neuronsku mrežu na MNIST skupu podataka. Na kraju, evaluirat ćemo modele na testnom skupu podataka i provesti njihovu usporedbu.

Ključne riječi

Klasifikacija, logistička regresija, metoda potpornih vektora, neuronska mreža

Comparison of traditional machine learning algorithms and neural network in classification problems

Summary

In this paper, we will examine classification tasks, describe some of the algorithms used for classification, and provide an overview of neural networks. Following that, we will implement and train models for logistic regression, support vector machine, and neural networks using the MNIST dataset. Finally, we will evaluate the models on a test dataset and compare them.

Keywords

Classification, logistic regression, support vector machine, neural network