»Mladi za napredek Maribora 2016«
33. srečanje

Preverjanje učinkovitosti naprednih modelov nevronskih mrež

Raziskovalno področje: računalništvo

Raziskovalna naloga

PROSTOR ZA NALEPKO

Avtor: URBAN DUH

Mentor: ANDREJ DUH

Šola: II. GIMNAZIJA MARIBOR

2016, Maribor

»Mladi za napredek Maribora 2016«
33. srečanje

Preverjanje učinkovitosti naprednih modelov nevronskih mrež

Raziskovalno področje: računalništvo

Raziskovalna naloga

PROSTOR ZA NALEPKO

2016, Maribor

KAZALO

KAZALO SLIK

KAZALO TABEL

KAZALO GRAFIKONOV

1. POVZETEK

V raziskovalni nalogi smo se ukvarjali z implementacijo in preizkušanjem algoritma linearne logistične regresije in nevronskih mrež na treh različnih naborih podatkov. Za vsak nabor podatkov smo izbrali optimalno arhitekturo nevronske mreže in optimalno vrednost regularizacijskega parametra in izračunali pričakovano natančnost. Nato smo izračunali še pričakovano natančnost algoritma linearne logistične regresije in ga primerjali s pričakovano natančnostjo algoritma nevronske mreže. Ugotovili smo, da sta prva dva nabora podatkov, na katerih smo preizkušali, manj kompleksna kot tretji in da uporaba algoritma nevronske mreže na njih ni smiselna, saj dosega podobne pričakovane natančnosti kot algoritem linearne logistične regresije, ki je hitrejši. Na tretjem naboru podatkov je bila uporaba algoritma nevronske mreže smiselna, saj je dosegala večje pričakovane natančnosti kot algoritem linearne logistične regresije. V zaključku smo prikazali uporabo modernejše knjižnice Deeplearning4j.

2. ZAHVALA

Zahvaljujemo se mentorju za pomoč in nasvete pri pisanju naloge ter možnosti uporabe zmogljivega računalniškega strežnika pri zadnjem poglavju eksperimentalnega dela naloge.

3. UVOD IN OPREDELITEV NALOGE

Idejo za raziskovalno nalogo smo dobili, ko smo brali članek na spletni strani revije Wired (Clark, L., 2015). Članek opisuje uporabo algoritmov umetne inteligence, ki jo je naredilo podjetje DeepMind Technologies. S pomočjo algoritmov umetne inteligence se program nauči igrati klasične računalniške igre iz konzole Atari 2600 brez predhodnega znanja. Programu podajo le informacije o slikovnih točkah na ekranu, možnost uporabe vseh različnih ukazov in cilj – doseči čim večje število točk. Po določenem številu poskusov se program nauči igrati igro ter dosega rezultate, ki so primerljivi ali celo boljši od človeških.

Slika 1. Ena izmed Atari 2600 iger (Clark, L., 2015).

Na spletu smo našli razlago, da takšni programi delujejo s pomočjo algoritma, ki se imenuje nevronska mreža. Njegovo delovanje se nam je zdelo zelo zanimivo, saj posnema delovanje človekovih možganov, torej se z raziskovanjem in izboljševanjem tega algoritma bližamo popolnemu posnemanju človeškega razmišljanja. Uporaben je za reševanje veliko različnih problemov, uporablja se na primer tudi v avtonomni vožnji, prepoznavanju človeške pisave, diagnosticiranju in napovedovanju bolezni itd.

Odločili smo se, da se bomo v raziskovalni nalogi ukvarjali s prikazom implementacije, uporabe in preizkušanjem algoritma nevronske mreže, ki ga bomo tudi primerjali z algoritmom linearne logistične regresije. V raziskovalni nalogi bomo:

- implementirali algoritem nevronske mreže in linearne logistične regresije,
- izbrali optimalno arhitekturo nevronske mreže na treh različnih bazah podatkov,
- izbrali optimalno vrednost regularizacijskega parametra na treh različnih bazah podatkov,
- primerjali natančnosti algoritma nevronske mreže in linearne logistične regresije.

V raziskovalni nalogi se bomo omejili na probleme klasifikacije.

4. TEORETIČNI DEL

4.1 Strojno učenje

Poznamo dve različni definiciji strojnega učenja. Arthur Samuel opisuje strojno učenje kot: »področje znanosti, ki omogoča računalnikom, da se učijo, brez da bi bili za to eksplicitno programirani.« (Ng, A., 2015, Week 1)
Drugo, bolj formalno definicijo je podal Tom Mitchell, ki pravi, da je strojno učenje »računalniški program, ki se uči iz izkušenj E, izvaja opravilo T in ima merilo uspešnosti P, kjer se njegova uspešnost pri opravilu T, merjeno s P, izboljšuje z izkušnjami E.« (Ng, A., 2015, Week 1)

Kot primer lahko predstavimo igranje igre iz konzole Atari, opisane v uvodu naloge. Izkušnje E so tam izkušnje, ki so bile pridobljene iz igranja igre, opravilo T je samo igranje igre, merilo uspešnosti P pa je število pridobljenih točk.

Strojno učenje delimo na dve različni področji: **nadzorovano** (angleško supervised) in **nenadzorovano** (angleško unsupervised) učenje. Pri nadzorovanem učenju imamo podan nabor podatkov (vhodov), za katere poznamo pripadajoče izhode. Nadzorovano učenje še naprej delimo na **regresijo** in **klasifikacijo**. Pri regresiji so izhodne spremenljivke zvezne, torej želimo poiskati takšno funkcijo, ki vhodnim spremenljivkam pripiše izhodne. Klasifikacija se ukvarja s problemi, kjer so izhodne spremenljivke diskretne, pri njej želimo vhodne podatke razvrstiti v diskretne kategorije (Witten, I. H., Frank, E. & Hall, M. A., 2011).

Pri nenadzorovanem učenju izhodov ne poznamo v naprej, vhodne podatke želimo na primer razdeliti v različne kategorije, ne da bi vedeli, kakšne so te kategorije.

Pri raziskovalni nalogi smo se osredotočili predvsem na probleme klasifikacije, vendar moramo za razumevanje le-teh poznati tudi algoritme regresije.

4.2 Linearna regresija

Linearna regresija je algoritem, ki želi naboru podatkov prilagoditi linearno funkcijo. Tej funkciji, ki vhodom pripiše izhode, pravimo **hipoteza**. Zapišemo jo lahko kot (Ng, A., 2015, Week 1):

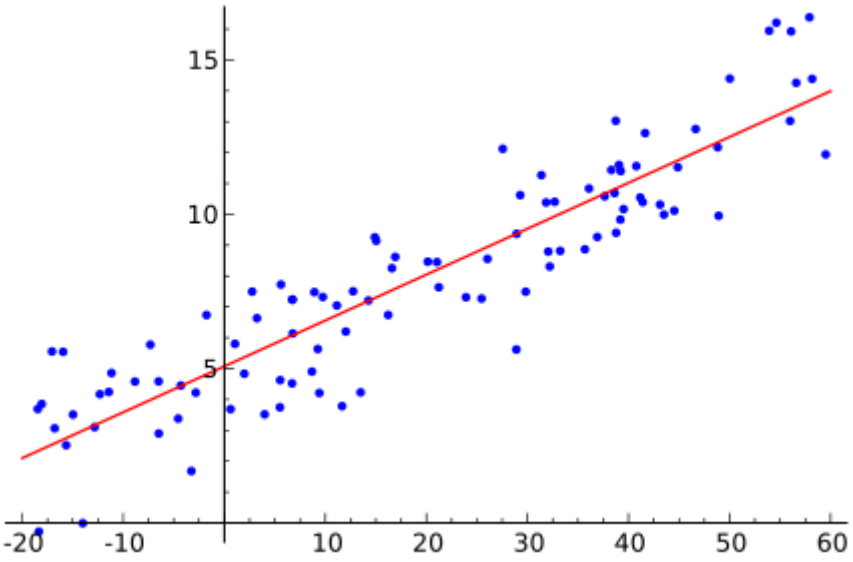
$$h_\theta(x_1, x_2, \dots x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Kjer so x_1, x_2, ... x_n vhodi ali **lastnosti** (angleško features), $\theta_1, \theta_2, \dots \theta_n$ neke konstante ali **uteži,** n pa je število različnih značilnosti oz. vhodov.

Napako hipoteze na nekem naboru podatkov izračunamo s **stroškovno funkcijo** (angleško cost function), ki je enaka (Ng, A., 2015, Week 1):

$$J(\theta_0, \theta_1, \dots \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta\left(x_1{}^{(i)}, x_2{}^{(i)}, \dots x_n{}^{(i)}\right) - y^{(i)})^2$$

Kjer je m enak številu parov vhodov in izhodov (v nadaljevanju **naborov lastnosti**), $y^{(i)}$ pričakovan izhod pri naboru lastnosti $x_1{}^{(i)}, x_2{}^{(i)}, \dots x_n{}^{(i)}$. Z $^{(i)}$ ne označujemo i-te potence, temveč i-ti par nabora lastnosti.



Slika 2. Primer grafa linearne regresije (Wikipedia, Linear Regression, 2016).

Na zgornji sliki vidimo primer grafa linearne regresije z eno lastnostjo. Na abscisni osi so nanašane lastnosti (x), na ordinatni pa izhodne vrednosti (y). Z modrimi pikami so označeni pari lastnosti in pričakovani izhodi, z rdečo črto pa hipoteza.

Če želimo poiskati hipotezo, ki se najbolje prilagaja našemu naboru podatkov, moramo najti takšne uteži, da je vrednost stroškovne funkcije J čim manjša. Poiskati moramo torej minimum funkcije $J(\theta_0, \theta_1 \dots \theta_n)$.
Iskanja minimuma funkcije se lahko lotimo na več načinov, na primer z algoritmom **gradientni spust** (angleško gradient descent), s knjižnicami v različnih jezikih itd. V raziskovalni nalogi bomo uporabljali predvsem funkcijo iz progama Octave *fminunc*, ki kot vhod zahteva funkcijo, ki računa matematično funkcijo, katere minimum želimo poiskati, in parcialne odvode po vseh spremenljivkah te funkcije. Parcialni odvod po θ_i nam pove, v katero smer moramo spreminjati parameter θ_i, da se bo vrednost funkcije J manjšala. Prav tako se vrednost parcialnega odvoda manjša, ko se bližamo minimumu funkcije. Avtorji te raziskovalne naloge sicer nismo vešči v računanju odvodov, vendar to za implementacijo linearne regresije ni pomembno, saj jih izračunamo po naslednjih formulah (Ng, A., 2015, Week 1):

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_1{}^{(i)}, x_2{}^{(i)}, \dots x_n{}^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_1{}^{(i)}, x_2{}^{(i)}, \dots x_n{}^{(i)}) - y^{(i)})\, x_i{}^{(i)}, \qquad kjer\ i \neq 0$$

Ko poiščemo minimum funkcije, dobimo takšne uteži, ki opisuje najbolje prilagojeno linearno funkcijo danemu naboru podatkov.

Linearna regresija ni uporabna le, ko so lastnosti in izhodi med seboj linearno povezani, saj lahko dodamo svoje lastnosti x_{n+1}, ki so na primer enake potencam ali korenom x_i (na primer $x_1{}^2$, $x_1{}^3$, $\sqrt{x_1}$ ...). Tako lahko dobimo različne nelinearne polinomske funkcije. Slabost tega je le, da moramo vedeti, kakšno funkcijo želimo prilagoditi naboru podatkov (Ng, A., 2015, Week 2).

4.3 Vektorizacija

Pisanje programov linearne regresije in drugih algoritmov strojnega učenja si zelo olajšamo, če uteži, lastnosti in izhode shranjujemo v matrikah. Z uporabo hitrih knjižnic linearne algebre zmanjšamo čas iskanje minimuma funkcije J, saj zmanjšamo potrebo po *for* zankah. V raziskovalni nalogi bomo shranjevali lastnosti v matriki X na naslednji način (Ng, A., 2015, Week 1):

$$X = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)} & \cdots & {x_n}^{(1)} \\ {x_0}^{(2)} & {x_1}^{(2)} & \cdots & {x_n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ {x_0}^{(m)} & {x_1}^{(m)} & \cdots & {x_n}^{(m)} \end{bmatrix}$$

Kjer je n enak številu lastnosti, m pa številu naborov lastnosti. ${x_a}^{(b)}$ označuje a-to značilnost v b-tem naboru lastnosti. Po dogovoru bodo vsi ${x_0}^{(i)}$ enaki 1.

Prav tako bomo pričakovane izhode shranjevali v matriki oziroma stolpičnem vektorju Y (Ng, A., Week 1):

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Označevanje je podobno kot v matriki X.

Uteži bomo shranjevali v matriki oziroma stolpičnem vektorju θ (Ng, A., 2015, Week 1):

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Če upoštevamo zgornje shranjevanje podatkov, lahko zapišemo vektorizirane enačbe za hipotezo, stroškovno funkcijo in parcialne odvode (Ng, A., 2015, Week 1):

$$h_\theta(X) = X \cdot \theta$$

$$J(\theta) = \frac{1}{2m} \ (h_\theta(X) - Y) \ .\hat{}\ 2$$

$$\frac{\partial}{\partial\theta_{0,1,\dots n}}J(\theta) = \frac{1}{m}X^T \cdot (h_\theta(X) - Y)$$

Z a .^ n označujem matriko n-tih potenc vsakega elementa matrike a in z a^T transponiranje matrike a. Pri $h_\theta(X)$ dobimo pripisane vrednosti shranjene v podobni matriki kot Y in pri $\frac{\partial}{\partial\theta_{0,1,\dots n}}J(\theta)$ dobimo matriko, ki ima na mestu $(i+1, 1)$ shranjen parcialni odvod $\frac{\partial}{\partial\theta_i}J(\theta_i)$.

4.4 Linearna logistična regresija

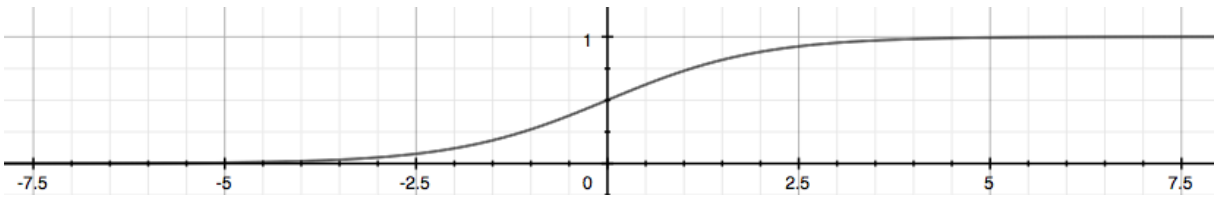
Algoritem linearne logistične regresije se uporablja za probleme **klasifikacije**, čeprav ima v imenu besedo regresija. Najprej bomo razložili logistično regresijo z dvema **razredoma**, to pomeni, da izhodi lahko zavzemajo dve različni vrednosti. Za ti dve vrednosti si bomo izbrali 1 in 0. Ponavadi ju imenujemo pozitivni (1) in negativni (0) razred.
Pri klasifikaciji želimo, da naša hipoteza napove **verjetnost**, da določen nabor lastnosti pripada nekemu razredu. Po dogovoru hipoteza napove verjetnost, da pripada pozitivnemu razredu (1) (Ng, A., Week 3):

$$h_\theta(x) = P(y = 1 \mid x;\ \theta)$$

Da to dosežemo, uporabimo sigmoidno funkcijo, ki jo bomo v raziskovalni nalogi označevali z $g(z)$. Sigmoidna funkcija zavzema vrednosti od 1 do 0 in jo zapišemo (Ng, A., 2015, Week 3):

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

Graf sigmoidne funkcije izgleda tako:



Slika 3. Graf sigmoidne funkcije (ML Wiki, 2016).

Lahko si torej mislimo, da sigmoidna funkcija spremenljivki z priredi vrednost iz intervala $(0, 1)$. Hipotezo pri logistični regresiji torej zapišemo kot (Ng, A., 2015, Week 3):

$$h_\theta(X) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n) = g(X \cdot \theta)$$

Uporabljen je vektoriziran zapis, ki je opisan v poglavju 4.3.

Pri logistični regresiji se moramo odločiti tudi za vrednost **praga odločitve**. Če hipoteza vrne vrednost, ki je večja ali enaka pragu odločitve, bo program predvidel, da ta nabor lastnosti pripada pozitivnemu razredu. Če pa hipoteza vrne vrednost, ki je manjša od praga odločitve, bo program predvidel, da ta nabor lastnosti pripada negativnemu razredu. Običajno za prag odločitve izberemo vrednost 0.5. To lahko zapišemo:

$$h_\theta(X) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(X) < 0.5 \rightarrow y = 0$$

Torej:

$$g(X \cdot \theta) \geq 0.5 \rightarrow y = 1$$
$$g(X \cdot \theta) < 0.5 \rightarrow y = 0$$

S stališča spremenljivke sigmoidne funkcije torej velja:

$$X \cdot \theta \geq 0 \rightarrow y = 1$$
$$X \cdot \theta < 0 \rightarrow y = 0$$

Za izbiranje optimalnih uteži potrebujemo tudi stroškovno funkcijo. Stroškovna funkcija, ki smo jo uporabili pri linearni regresiji, pri logistični regresiji ni primerna, saj ob upoštevanju nove hipoteze ne bi bila konveksna. To bi pomenilo, da bi program dosti težje poiskal minimum funkcije in s tem optimalne uteži. Zaradi tega uporabimo malo drugačno stroškovno funkcijo, ki pa je konveksna (Ng, A., 2015, Week 3):

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \log\left(h_\theta\left(X^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(X^{(i)}\right)\right)\right) =$$

$$= -\frac{1}{m}\left(\log\left(h_\theta(X)\right)^T \cdot Y + \log\left(1 - h_\theta(X)\right)^T \cdot (1 - Y)\right)$$

Parcialni odvod te stroškovne funkcije po vseh utežeh je popolnoma enak kot parcialni odvod stroškovne funkcije pri linearni regresiji, zato ga bomo ponovno zapisali le v vektorizirani obliki (Ng, A., 2015, Week 3):

$$\frac{\partial}{\partial\theta_{0,1,\dots n}} J(\theta) = \frac{1}{m} X^T \cdot (h_\theta(X) - Y)$$

Ker znamo zapisati stroškovno funkcijo in njene parcialne odvode, lahko optimalne uteži poiščemo na povsem enak način kot pri linearni regresiji. Podobno kot pri linearni regresiji lahko dodamo tudi kvadratne, kubične, korenske itd. lastnosti, vendar ima to povsem enake pomanjkljivosti kot enakovreden postopek pri linearni regresiji.

4.4.1 Metoda eden proti vsem

Pogosto se srečamo tudi s problemi, kjer y lahko pripada k različnim razredom, ki jih oštevilčimo s naravnimi števili od 0 do $k - 1$. Takrat ne moremo uporabiti klasične logistične regresije, temveč uporabimo metodo **eden proti vsem** (angleško one vs. all). Takrat definiramo k različnih hipotez, kjer vsaka izraža verjetnost, da nek nabor lastnosti pripada enemu razredu (Ng, A., 2015, Week 3).

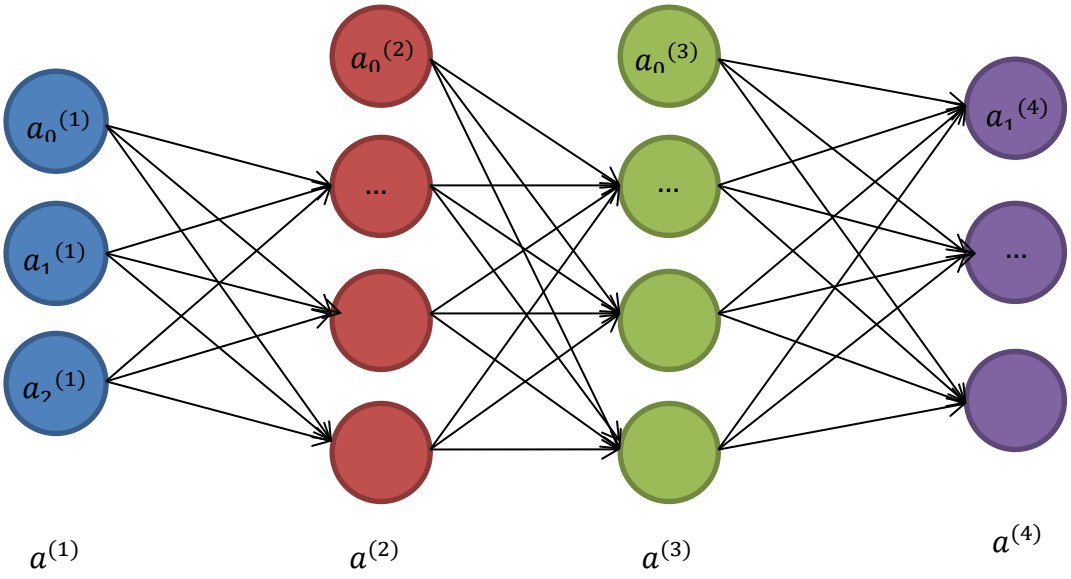
$$h_\theta{}^{(0)}(x) = P(y = 0 \mid x;\ \theta^{(0)})$$
$$h_\theta{}^{(1)}(x) = P(y = 1 \mid x;\ \theta^{(1)})$$
$$\dots$$
$$h_\theta{}^{(k-1)}(x) = P(y = k - 1 \mid x;\ \theta^{(n)})$$

Z $^{(m)}$ ponovno ne označujemo potence, temveč zaporedno število hipoteze in uteži.

Vsem hipotezam z že opisanimi metodami poiščemo optimalne uteži in ob klasificiranju novega nabora lastnosti izračunamo vrednosti vseh hipotez. Program nato predpostavi, da nabor pripada razredu tiste hipoteze, pri kateri je izračunal največjo verjetnost.

4.5 Nevronske mreže

Linearna logistična regresija ni primerna za reševanje problemov, kjer bi se naboru podatkov najbolje prilagodila nelinearna hipoteza. Problem je, da običajno izraza za hipotezo ne poznamo vnaprej. Tako moramo upoštevati vse kombinacije zmnožkov, kvadratov itd. med vsemi lastnostmi, kar pa nam lahko vzame veliko časa. Veliko lažjo rešitev za nelinearne hipoteze nam ponujajo algoritmi nevronskih mrež. Nevronske mreže delujejo podobno kot naši možgani, predstavljamo si jih lahko tako:



Slika 4. Shema nevronske mreže (Ng, A., 2015, Week 4).

S krogom je na shemi označen en **nevron** ali **vozlišče** (angleško node), ki so združeni v nivoje. Na shemi predstavljajo en nivo vsi nevroni pobarvani z enako barvo. Nevrone označujemo z $a_m{}^{(n)}$, kjer n ustreza zaporednemu številu nivoja, m pa zaporednemu število nevrona v nivoju. Nevron z $m = 0$ ima vedno vrednost 1 in mu pravimo **bias**. Prvemu nivoju pravimo **vhodni nivo** (angleško input layer), zadnjemu **izhodni nivo** (angleško output layer), vsem ostalim pa **skriti nivoji** (angleško hidden layers). Na vsakem nevronu poteka logistična regresija, lastnosti logistične regresije na enem nevronu so vse vrednosti nevronov iz prejšnjega nivoja. Vsak nivo ima tudi svoje uteži, ki so na shemi predstavljene s puščicami. Označimo jih s $\theta_{x,y}{}^{(n)}$, kjer n ustreza zaporednemu številu nivoja, x zaporednemu številu nevrona, katerega vrednost računamo, y pa zaporednemu številu nevrona, katerega vrednost pomnožimo z vrednostjo te uteži. Število nevronov v vhodnem nivoju je vedno za ena večje od števila lastnosti, saj je $a_0{}^{(1)}$ vedno enak 1. Število nevronov v izhodnem nivoju je vedno

enako številu razredov, v katere spadajo nabori lastnosti v našem naboru podatkov. Izhodni nevron z zaporednim številom n izraža verjetnost, da nabor značilnosti spada v n-ti razred. Program naj za nabor lastnosti predvidi razred, kateremu pripada največja verjetnost na izhodnem nivoju (Ng, A., 2015, Week 4).

$$a_1{}^{(L)} = P(y = 0 \mid x;\ \theta^{(L)})$$
$$a_2{}^{(L)} = P(y = 1 \mid x;\ \theta^{(L)})$$
$$\dots$$
$$a_K{}^{(L)} = P(y = K - 1 \mid x;\ \theta^{(L)})$$

Kjer je K enak številu vseh razredov, v katere pripadajo nabori lastnosti, L pa številu vseh nivojev.

Računanje vrednosti vseh nevronov zapišemo tako (Ng, A., 2015, Week 4):

$$a_0{}^{(1)} = 1$$
$$a_1{}^{(1)} = x_1$$
$$\dots$$
$$a_1{}^{(2)} = g(\theta_{1,0}{}^{(2)} a_0{}^{(1)} + \theta_{1,1}{}^{(2)} a_1{}^{(1)} + \dots + \theta_{1,n}{}^{(2)} a_n{}^{(1)})$$
$$a_2{}^{(2)} = g(\theta_{2,0}{}^{(2)} a_0{}^{(1)} + \theta_{2,1}{}^{(2)} a_1{}^{(1)} + \dots + \theta_{2,n}{}^{(2)} a_n{}^{(1)})$$
$$\dots$$
$$a_k{}^{(j)} = g(\theta_{k,0}{}^{(j)} a_0{}^{(j-1)} + \theta_{k,1}{}^{(j)} a_1{}^{(j-1)} + \dots + \theta_{1,n}{}^{(j)} a_n{}^{(j-1)}), kjer\ je\ j > 1$$

Vse vrednosti $a^{(i)}$ shranjujemo v matriki A_i (Ng, A., Week 4):

$$A_i = \begin{bmatrix} a_1{}^{(i),(1)} & a_2{}^{(i),(1)} & \dots & a_n{}^{(i),(1)} \\ a_1{}^{(i),(2)} & a_2{}^{(i),(2)} & \dots & a_n{}^{(i),(2)} \\ \vdots & \vdots & & \vdots \\ a_1{}^{(i),(m)} & a_2{}^{(i),(m)} & \dots & a_n{}^{(i),(m)} \end{bmatrix}$$

Kjer $a_n{}^{(i),(m)}$ predstavlja nevron v i-tem nivoju, z zaporedno številko n, v naboru lastnosti m. Z $A_{i,1}$ označimo matriko A_i, ki ima na levi strani dodan stolpec, kjer so vse vrednosti enake 1 (to potrebujemo za bias nevrone).

Podobno shranimo tudi uteži (Ng, A., 2015, Week 4):

$$\theta_i = \begin{bmatrix} {\theta_{1,0}}^{(i)} & {\theta_{1,1}}^{(i)} & \dots & {\theta_{1,n}}^{(i)} \\ {\theta_{2,0}}^{(i)} & {\theta_{2,1}}^{(i)} & \dots & {\theta_{2,n}}^{(i)} \\ \vdots & \vdots & & \vdots \\ {\theta_{k,0}}^{(i)} & {\theta_{k,1}}^{(i)} & \dots & {\theta_{k,n}}^{(i)} \end{bmatrix}$$

S pomočjo tega lahko vektoriziramo računanje vrednosti nevronov :

$$A_{1,1} = X$$
$$A_2 = g(A_{1,1} \cdot {\theta_1}^T)$$
$$\dots$$
$$A_k = g(A_{k-1,1} \cdot {\theta_{k-1}}^T)$$

4.5.1 Stroškovna funkcija in parcialni odvodi pri nevronski mreži z regularizacijo

Če želimo uporabiti v tej nalogi opisane postopke za iskanje optimalnih uteži, potrebujemo stroškovno funkcijo in parcialne odvode le-te po vseh vrednostih uteži. Stroškovno funkcijo zapišemo takole (Ng, A., Week 5):

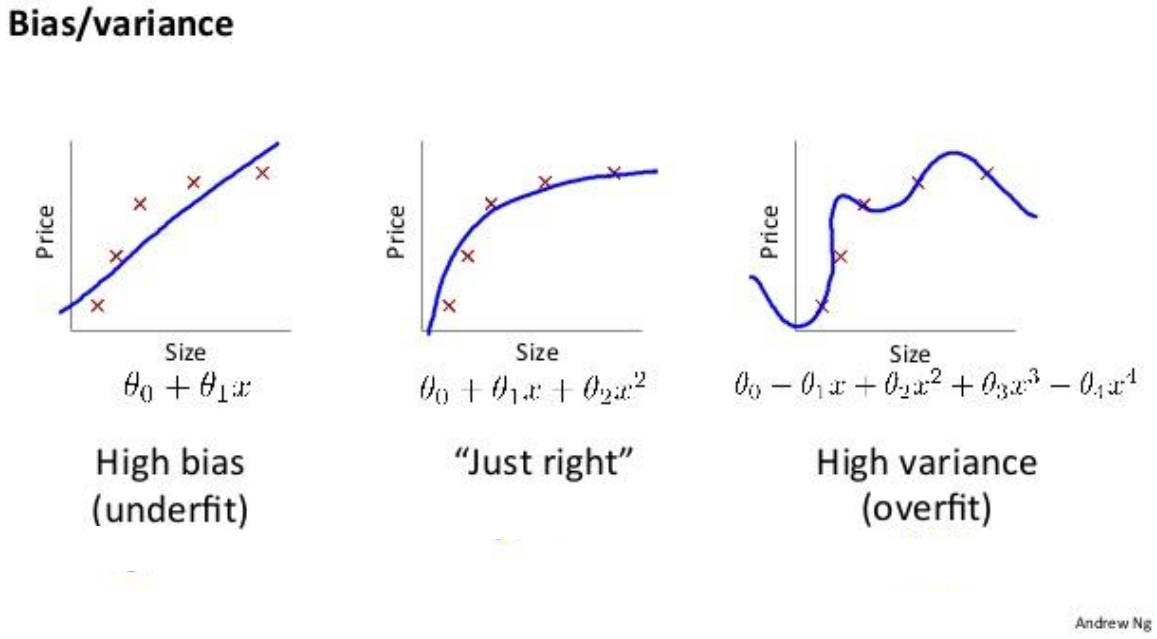
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left({y_k}^{(i)}\log\left({a_k}^{(L),(i)}\right) + \left(1-{y_k}^{(i)}\right)\log\left(1-{a_k}^{(L),(i)}\right)\right) + \\ + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}({\theta_{j,i}}^{(l)})^2$$

Kjer je ${y_k}^{(i)}$ enak pričakovani vrednosti nevron ${a_k}^{(L),(i)}$, L enak številu nivojev, s_l enak številu nevornov v nivoju l (brez bias nevrona), K številu razredov, v katere spadajo nabori lastnosti (kar je tudi enako številu nevronov v izhodnem nivoju), λ pa **regularizacijskemu parametru**. Vektorizirane oblike te enačbe v teoretičnem delu ne bomo zapisali, saj je v literaturi nismo našli. Ta stroškovna funkcija sicer ni konveksna, vendar to v večini primerov ne povzroča velikih težav (Ng, A., 2015, Week 5).

Opazimo, da je prvi člen stroškovne funkcije zelo podoben stroškovni funkciji linearne logistične regresije. Drugemu členu pravimo **regularizacijski člen** in skrbi, da naša nevronska mreža nima velike **variance** (angleško high variance). To pomeni, da se naša hipoteza oziroma nevronska mreža zelo dobro prilagodi že znanim podatkom, vendar slabo

predvidi razred novemu naboru lastnosti. Do tega običajno pride zato, ker je naša hipoteza preveč kompliciran in naredi veliko nepotrebnih »zavojev«. Na drugi strani imamo velik **bias** (angleško high bias), kar pomeni, da je naša hipoteza preveč enostavna in se podatkom ne prilagodi dobro. Regularizacijski parameter λ izraža stopnjo regularizacije, večji kot je, močneje bomo regularizirali. Izbira optimalne vrednosti regularizacijskega parametra je lahko težavna, saj preveč majhna vrednost povzroči veliko varianco, preveč velika pa velik bias (Ng, A., Week 3).

Regularizacijo lahko uporabimo tudi pri linearni regresiji in linearni logistični regresiji, vendar je v tej nalogi ne bomo, saj je smiselna le ob uporabi polinomskih členov (lastnosti). Na sliki 5 lahko vidimo hipotezo linearne regresije z eno lastnostjo in z dodanimi polinomskimi členi. Na grafih so na abscisno os nanešene lastnosti (x), na ordiantno pa vrednosti izhodov. Z rdečimi križci so označeni pari lastnosti in izhodov, z modro črto pa vrednosti hipoteze pri optimalnih utežeh.



Slika 5. Hipoteza z visokim biasom in hipoteza z visoko varianco (Ng. A, 2015, Week 3).

Zaradi zelo kompleksne nelinearne hipoteze je računanje parcialnih odvodov pri nevronski mreži precej težavno, zato se poslužujemo posebnih numeričnih algoritmov. V tej raziskovalni nalogi bomo uporabljali **algoritem vzvratnega razširjanja**, ki temelji na tem, da najprej izračunamo vrednosti $\delta_j^{(l)(t)}$, ki izražajo »napako« nevrona z zaporednim številom j v

nivoju l, v naboru značilnosti t. Izračunamo jih po naslednjih enačbah (Ng, A., 2015, Week 5):

$$\delta_j{}^{(L)(t)} = a_j{}^{(L)(t)} - y_j{}^{(t)}$$

$$\delta_j{}^{(l)(t)} = \left(\theta_{1,j}{}^{(l)}\, \delta_1{}^{(l+1)(t)} + \theta_{2,j}{}^{(l)}\, \delta_2{}^{(l+1)(t)} + \cdots + \theta_{s(l+1),j}{}^{(l)}\, \delta_{s(l+1)}{}^{(l+1)(t)}\right) a_j{}^{(l)(t)}\left(1 - a_j{}^{(l)(t)}\right), \qquad kjer\ l < L$$

Parcialne odvode po vseh utežeh nato izračunamo na naslednji način (Ng, A., 2015, Week 5):

$$\frac{\partial}{\partial \theta_{i,0}{}^{(l)}} J(\theta) = \frac{1}{m}\left(\sum_{t=0}^{m} a_0{}^{(l)(t)} \delta_i{}^{(l+1)(t)}\right)$$

$$\frac{\partial}{\partial \theta_{i,j}{}^{(l)}} J(\theta) = \frac{1}{m}\left(\sum_{t=0}^{m} a_j{}^{(l)(t)} \delta_i{}^{(l+1)(t)} + \lambda \theta_{i,j}{}^{(l)}\right), \qquad kjer\ j \neq 0$$

Algoritem vzvratnega razširjanja lahko tudi matematično dokažemo, vendar za to avtorji te raziskovalne naloge nimamo dovolj matematičnega znanja. Dokaz prav tako ni pomemben za implementacijo algoritma. Prav tako lahko vzvratno razširjanje tudi vektoriziramo, vendar tega nismo zasledili v naših virih.

S tem znanjem lahko uporabimo enega izmed algoritmov za minimiziranje vrednosti funkcije, da poiščemo optimalne uteži. Pogosto pa ti algoritmi zahtevajo, da podamo neko začetno vrednost uteži. Pri nevronskih mrežah je pomembno, da začetne vrednosti uteži niso vse enake, saj bi ob uporabi vzvratnega razširjanja vse uteži ostale med seboj enake. To rešimo tako, da vse uteži v nivoju l naključno inicializiramo na vrednost znotraj intervala $[-\epsilon_l, \epsilon_l]$, ϵ_l ponavadi nastavimo na vrednost (ML Wiki, 2016):

$$\epsilon_l = \frac{\sqrt{6}}{\sqrt{(s_l + 1) + s_{l+1}}}$$

4.6 Normalizacija

Pogosto se nam zgodi, da je v nekem naboru podatkov obseg vrednosti ene lastnosti mnogo večji kot obseg vrednosti druge lastnosti. Na primer prva lastnost zavzema vrednosti na

intervalu $[-1000, 1000]$, druga pa $[0{,}001, 0{,}002]$. V takšnih primerih bo iskanje minimuma stroškovne funkcije vzelo veliko več časa in bo manj natančno, kot če bi bile vse lastnosti v približno enakem obsegu (Ng, A., 2015, Week 2 in Bishop, C. M., 2006). Zaradi tega je smiselno lastnosti **normalizirati**, spremeniti obseg na približno enako vrednost pri vseh častnostih. To naredimo z naslednjo enačbo (Ng, A., 2015, Week 2):

$$x_i^{(j)} = \frac{x_i^{(j)} - \mu_i}{s_i}$$

Kjer je μ_i aritmetična sredina vseh i-tih lastnosti v naboru podatkov in s_i najmanjša vrednost i-te lastnosti v naboru podatkov odšteta od največje vrednosti i-te lastnosti v naboru podatkov.

4.7 Računanje natančnosti

Pogosto želimo za nek nabor podatkov izračunati natančnost neke hipoteze, kar lahko naredimo le, če pričakovane izhode poznamo. To naredimo tako, da z hipotezo izračunamo izhode danim lastnostim in jih primerjamo s pričakovanimi izhodi. Natančnost (acc) dobimo tako, da seštejemo število naborov lastnosti, za katere smo pravilno predvideli izhode (n_p) in jih delimo s celotnim številom naborov lastnosti, ki smo jih uporabili (m):

$$acc = \frac{n_p}{m}$$

Če natančnost pomnožimo s 100, dobimo število, ki izraža, v koliko odstotkih lahko pričakujemo, da bo program prav predvidel izhod (Ng, A., 2015, Week 6).

5. PRAKTIČNI DEL

5.1 Metodologija

Praktični del raziskovalne naloge lahko v grobem razdelimo na tri sklope. V prvem smo se ukvarjali z implementacijo in preizkušanjem algoritma logistične regresije. V drugem smo napisali program za izvajanje algoritma nevronskih mrež, ga preizkusili in primerjali z algoritmom logistične regresije. V zadnjem, tretjem sklopu smo želeli preizkusiti eno izmed modernejših knjižnic za strojno učenje. Odločili smo se za knjižnico Deeplearning4j.

Vse programe za prve dva dela praktičnega dela smo pisali v jeziku Octave, saj ima enostaven zapis, hkrati pa mogoče zelo hitro računanje z matrikami in vektorji. Prav tako ima že implementirane funkcije za iskanje minimuma matematičnih funkcij, kar je za strojno učenje zelo ugodno. Programi, napisani v jeziku Octave, sicer niso tako učinkoviti in hitri, kot v nekaterih drugih jezikih (na primer C, C++, Java), vendar to za izvedbo raziskovalne naloge ni bilo tako pomembno. Imena funkcij bomo v raziskovalni nalogi pisali v ležeči pisavi ter z uklepajem in zaklepajem na koncu imena funkcije. Ko bomo želeli razložiti ali poudariti argumente, ki jih določena funkcija zahteva, jih bomo pisali v oklepajih na koncu imenu ter jih ločili z vejico.

Algoritma logistične regresije in nevronske mreže smo preverili na treh različnih naborih podatkov. Prvi je klasifikacija rastlin Iris Setosa, Iris Versicolour in Iris Virginica (perunike). Nabor podatkov vključuje 150 naborov lastnosti z 4 lastnostmi. Nabor je en izmed najbolj znanih naborov podatkov v strojnem učenju in tudi en izmed najbolj enostavnih. Drugi nabor podatkov je klasifikacija treh različnih tipov italijanskih vin. Nabor podatkov vključuje 178 naborov lastnosti s 13 lastnostmi. Tretji nabor podatkov je diagnosticiranje tumorjev na prsih. Vključuje dva različna razreda, maligni in benigni tumor, 569 naborov lastnosti z 30 lastnostmi. Vse nabore podatkov smo pridobili iz spletne strani Machine Learning Repository (Lichman, M., 2013) in jih najdemo pod ključnimi besedami Iris, Wine in Breast Cancer Wisconsin (Diagnostic) ter jih uredili tako, da so posamezne lastnosti ločene samo s presledki.

5.2 Implementacija linearne logistične regresije

Ker smo želeli primerjati učinkovitost nevronskih mrež z drugimi algoritmi, smo morali implementirati tudi te algoritme. Odločili smo se za linearno logistično regresijo, ker smo jo spoznali med učenjem algoritma nevronske mreže. Vse potrebne datoteke za logistično regresijo so shranjene v mapi Logistic Regression.

Najprej smo napisali sigmoidno funkcijo. To je precej enostavno, saj moramo v programskem jeziku napisati le predpis za to funkcijo. Poimenovali smo jo *sigmoid(z)* in vrne vrednost sigmoidne funkcije za spremenljivko z.

Naslednja funkcija, ki jo potrebujemo za linearno logistično regresijo, je funkcija, ki vrne vrednost stroškovne funkcije in parcialne odvode po vseh utežeh. Poimenovali smo jo *costFunction(theta, X, Y)*, ki vrne spremenljivki J in $grad$. $Theta$ je vektor uteži (njegova ureditev je opisana v poglavju 4.3), X je matrika vseh lastnosti, Y je matrika pričakovanih izhodov, J je vrednost stroškovne funkcije in $grad$ je vektor, ki smo ga v teoretičnem delu označevali s $\frac{\partial}{\partial \theta_{0,1,\dots n}} J(\theta)$. Implementacija te funkcije prav tako ni povzročala problemov, saj je le zapisovanje že izpeljanih obrazcev v programskem jeziku.

Nato smo se lotili pisanja funkcije, ki za neko začetno vrednost uteži in nabor podatkov vrne optimalne vrednosti uteži, brez uporabe metode eden proti vsem. Poimenovali smo jo *train(initial_theta, X, Y)*, in vrne spremenljivko $theta$. $Initial\_theta$ je vektor začetnih vrednosti uteži, X je matrika vseh lastnosti, Y je vektor pričakovanih izhodo in $theta$ vektor optimalnih uteži. Za iskanje minimuma stroškovne funkcije in s tem optimalnih vrednosti uteži smo uporabili že implementirano funkcijo programskega jezika Octave *fminunc()***.** Kot prvi argument smo ji podali funkcijo *@(t) costFunction(t, X, Y)*, kar omogoča funkciji *fminunc()* klicanje funkcije *costFunction()* samo z njenim prvim argumetov, saj bosta X in Y v vseh klicih funkcije konstantna. Kot drugi argument smo ji podali $initial\_theta$, in kot zadnji skalarno strukturo tipa *optimset*, ki funkciji *fminunc()* sporoči, da *costFunction()* račun tudi odvode po vseh spremenljivkah, in nastavi največje število iteracij na 400, s čimer se izognemo preveč dolgemu računanju minimuma.

Naslednji dve funkciji sta bili precej enostavni. Funkcija *predict(theta, X)* vrne vrednost hipoteze, če je $theta$ enak vektorju uteži in X matriki naborov lastnosti. Funkcija *normalize(X)* vrne nabor podatkov X, normaliziran po metodi, opisani v teoretičnem delu.

Na koncu smo se lotili še pisanje zadnjih dveh zelo podobnih funkcij, ki za nabor podatkov poiščeta optimalne uteži in izračunata pričakovano natančnost hipotez z uporabo metode eden proti vsem. Poimenovali smo ju *buildOne(database)* in *buildOneVsAll3Class(database),* obe vrneta matriko optimalnih uteži $theta$ za vse hipoteze, in pričakovano natančnost acc. Spremenljivka $database$ mora biti matrika, ki ima v vsaki vrstici na prvih mestih shranjen nabor lastnosti, na zadnjem pa pričakovane izhode. Njuna edina razlika je, da prva deluje za nabor podatkov, ki spadajo v 2 razreda (brez metode eden proti vsem), druga pa za nabor podatkov, ki se klasificirajo v 3 razrede in uporabi metodo eden proti vsem. Če želimo izračunati pričakovano natančnost hipoteze, moramo nabor podatkov najprej razdeliti na dve skupini. Nato s prvo skupino poiščemo optimalne vrednosti uteži in s pridobljenimi utežmi izračunamo natančnost v drugi skupini. Ponavadi damo v prvo skupino okoli 80% celotnega nabora podatkov, v drugega pa 20% (Ng, A., 2015, Week 6). Prav tako je pomembno, da ves nabor podatkov najprej naključno premešamo (seveda ohranimo pripadajoče lastnosti skupaj, premešamo le vrstni red naborov lastnosti), saj s tem dosežemo približno enako pogostost razredov v obeh skupinah. Funkciji si pri izračunu optimalnih uteži za vse hipoteze pomagata z že prej napisano funkcijo *train()*, pri računanju pričakovane natančnosti pa z funkcijo *predict().*

5.2.1 Natančnost algoritma linearne logistične regresije

Z uporabo funkcij *buildOneVsAll3Class()* (pri prvih dveh naborih podatkov) in *buildOne()* (pri tretjem naboru podatkov) smo izračunali povprečno pričakovano natančnost algoritma logistične regresije na posameznih naborih podatkov. Preden smo uporabili funkcije smo z ukazom *load* shranili nabore podatkov v matrike $IRIS\_urejeno$, $WINE\_urejeno$ in $CANCER\_urejeno$. Za vsak nabor podatkov smo funkcijo zagnali deset krat in rezultate vpisali v spodnjo tabelo. Klic funkcije za 1. nabor podatkov se je glasil *[theta, acc] = buildOneVsAll3Class(normalize(IRIS_urejeno)),* kjer se je v spremenljivko acc shranila pričakovana natančnost, v spremenljivko $theta$ pa optimalen vektor uteži. Klic funkcije za 2.

nabor podatkov se je glasil *[theta, acc] = buildOneVsAll3Class(normalize([WINE_urejeno(:, 2:end), WINE_urejeno(:, 1)]))*, kjer se je v spremenljivko acc shranila pričakovana natančnost, v spremenljivko $theta$ pa optimalen vektor uteži. Klic funkcije za 3. nabor podatkov se je glasil *[theta, acc] = buildOne(normalize([CANCER_urejeno(:, 3:end), CANCER_urejeno(:, 2)]))*, kjer se je v spremenljivko acc shranila pričakovana natančnost, v spremenljivko $theta$ pa optimalen vektor uteži. Pri zadnjih dveh naborih podatkov smo morali spremeniti obliko shranjevanja podatkov, saj se način shranjevanja podatkov v datotekah ne ujema s shranjevanjem podatkov, ki ga pričakujeta funkciji. Pri tretjem naboru podatkov smo izpustili prvi stolpec, saj ta vsebuje zaporedno število pacienta, ki ne vpliva na diagnosticiranje tumorja.

Tabela 1. Natančnosti logistične regresije na različnih naborih podatkov (lasten arhiv).

	1. nabor podatkov	2. nabor podatkov	3. nabor podatkov
1.	0,933	1	0,974
2.	0,967	0,972	0,982
3.	1	1	0,974
4.	0,967	1	0,965
5.	1	0,944	0,991
6.	1	0,972	0,947
7.	0,967	0,972	0,974
8.	1	0,944	0,991
9.	0,933	1	0,947
10.	0,967	0,972	0,965
Povprečje:	**0,973 ± 0,04**	**0,978 ± 0,034**	**0,971 ± 0,024**

Iz tabela je razvidna zelo visoka natančnost algoritma linearne logistične regresije, povprečno celo okoli 97,4%. To ni bilo v skladu z našimi pričakovanji, takšno natančnost smo pričakovali pri algoritmu nevronskih mrež, saj smo menili, da bo logistična regresija preveč enostaven algoritem za naše nabore podatkov.

Pomemben podatek je tudi največja vrednost pričakovane natančnosti. Če bi želeli z izračunanimi utežmi reševati dejanske probleme, bi izbrali model, ki ima največjo

pričakovano natančnost. Pri prvem modelu je bila največja pričakovana natančnost 1, pri drugem 1 in pri tretjem 0,991. Iz tega lahko predpostavljamo, da je tretji nabor podatkov bolj kompleksen kot prva dva. To je res, saj tretji nabor podatkov vsebuje 30 lastnosti, prva dva pa le 4 in 13. Prav tako je diagnosticiranje tumorja (tretji nabor podatkov) bolj zapleteno kot klasificiranje rož (prvi nabor podatkov) ali vrst vina (drugi nabor podatkov).

5.3 Implementacija algoritma nevronskih mrež

Za implementacijo algoritma nevronski mrež smo uporabili dve že napisani funkciji; *sigmoid()* in *normalize()*. Vse datoteke povezane z nevronskimi mrežami so shranjene v mapi Neural Network.

Najprej smo se morali odločiti, kakšno arhitekturo nevronskih mrež bomo uporabili. Ker naše baze podatkov niso preveč kompleksne in ker smo bili omejeni zaradi moči našega računalnika, smo se odločili za tri- in štirinivojsko nevronsko mrežo. Število nevronov v posameznem nivoju nismo izbrali vnaprej, saj smo program napisali tako, da lahko to nastavimo ob različnih klicih funkcije. To je bilo veliko lažje, kot pisati funkcijo, ki deluje za različna števila nivojev, saj število nevronov spremeni le velikost matrik, število nivojev pa spremeni potek algoritma za izračunavanje izhodov in algoritma vzvratnega razširjanja.

Podobno kot pri implementaciji logistične regresiji smo nato napisali funkcijo, ki računa vrednost stroškovne funkcije in parcialne odvode stroškovne funkcije po vseh utežeh. Tokrat smo imeli funkciji dve, eno za trinivojsko nevronsko mrežo (en skrit nivo) in eno za štirinivojsko nevronsko mrežo (dva skrita nivoja). Imenovali smo ju *costFunction1(thetavec, x, y, lambda, input_layer_size, hidden_layer_size, num_labels)* in *costFunction2(thetavec, x, y, lambda, input_layer_size, hidden_layer1_size, hidden_layer2_size, num_labels)*. V vektroju $thetavec$ so zapisane vse vrednosti uteži, najprej prva vrstica matrike $\theta^{(1)}$, nato druga vrstica matrike $\theta^{(1)}$ ... nato prva vrstica matrike $\theta^{(2)}$ itd. Ta vektor na začetku same funkcije spremenimo v matrike, pri čemer nam pomagajo spremenljivke $input\_layer\_size, hidden\_layer1\_size, hidden\_layer2\_size$ in num_labels, v katere po vrsti shranimo velikosti vhodnega, prvega skritega, drugega skritega in izhodnega nivoja. Pri funkciji *costFunction1()* imamo seveda samo en skrit nivo in s tem tudi samo eno velikost

skritega nivoja. Argumenta x in y sta podobno kot pri logistični regresiji matriki lastnosti in pričakovanih izhodov, $lamdba$ pa regularizacijski parameter. Obe funkciji vrneta spremenljivki J, ki predstavlja vrednost stroškovne funkcije, in $grad$, ki predstavlja vektor parcialnih odvodov po vseh utežeh in je urejen podobno kot $thetavec$. Zakaj je smiselno podajati odvode in uteže v vektorjih namesto v matrikah bomo pojasnili v naslednjih odstavkih.

Po spremembi vektorja uteži v matrike, smo izračunali vrednosti vseh nevronov po formulah, ki smo jih zapisali v teoretičnem delu. Računanje stroškovne funkcije in izvedba algoritma vzvratnega razširjanja pa ni bila tako lahka, saj v virih nismo zasledili vektoriziranega zapisa teh dveh operacij. Po razmisleku in ob pomoči mentorja smo napisali program, ki te dve operaciji računa vektorizirano. Večje probleme je povzročal predvsem algoritem vzvratnega razširjanja, kjer smo vrednosti $\delta_j^{(l)}$ (poglavje 4.5.1) shranjevali v matrike $deltal$ (namesto črke l je vstavljeno zaporedno število nivoja) na podoben način kot vrednosti posameznih nevronov.

Po uspešno napisani funkcijah za računanje stroškovnih funkcij smo se lotili funkcij, ki za nek nabor podatkov izračunata optimalne vrednosti uteži. Imenovali smo ju *train1(initial_theta1, initial_theta2, x, y, lambda)* in *train2(initial_theta1, initial_theta2, initial_theta3, x, y, lambda)* in vrneta spremenljivke $theta1$ in $theta2$ oz. $theta1$, $theta2$ in $theta3$. V matrikah $initial\_theta$ so shranjene začetne vrednosti uteži, v matrikah x in y lastnosti in pričakovani izhodi, v $lambda$ vrednost regularizacijskega parametra in v matrikah $theta$ izračunane optimalne vrednosti uteži.

Računanja optimalnih vrednosti uteži smo se ponovno lotili z funkcijo *fminunc*, ki smo ji kot prvi argument podali funkcijo *costFunction1()* oz. *costFunction2()* na podoben način kot pri linearni logistični regresiji, kot drugi argument vektor začetnih vrednosti uteži in kot tretji argument enako strukturo tipa *optimset* kot pri linearni logistični regresiji. Zraven tega funkcija matrike $initial\_theta$ najprej spremeni v vektor, po izračunanem vektroju optimalnih vrednosti uteži pa nazaj v matrike.

Funkciji *predict1(theta1, theta2, x)* in *predict2(theta1, theta2, theta3, x)* sta precej enostavni. Za vrednosti uteži shranjenih v matrikah $theta$ in matriko lastnosti x vrneta matriko vrednosti

nevronov v izhodnjem nivoju p. Vektoriziran zapis teh operacij je opisan v teoretičnem delu naloge.

Nato smo se lotili funkcij *build1(database, hidden_layer_size, lambda)* in *build2(database, hidden_layer1_size, hidden_layer2_size, lambda).* Te funkciji za nabor podatkov $database$, velikosti skritih nivojev $hidden\_layer\_size$ oz. $hidden\_layer1\_size$ in $hidden\_layer2\_size$ ter regularizacijskih parameter $lambda$ vrneta optimalne vrednosti uteži $theta1$ in $theta2$ oz. $theta1, theta2$ in $theta3$. Veliko dela je opravljenega že v funkcijah *train1()* in *train2()*, funkciji morata le še nabor podatkov ločiti na matriko naborov lastnosti x in matriko pričakovanih izhodov y ter naključno inicializirati začetne vrednosti uteži. Postopek naključnega inicializiranja uteži je opisan v teoretičnem delu raziskovalne naloge.

Na koncu smo napisali še funkcijo, ki izračuna optimalne uteži za tri- in štirinivojsko nevronsko mrežo ter njuno pričakovano natančnost. Poimenovali smo jo *buildAndCompareNN(database, hidden_layer_3_1_size, lambda1, hidden_layer_4_1_size, hidden_layer_4_2_size, lambda2),* kjer je $database$ nabor podatkov, $hidden\_layer\_3\_1\_size$ velikost skritega nivoja v trinivojski nevronski mreži, *hidden_layer_4_1_size* in *hidden_layer_4_2_size* velikosti skritih nivojev v štirinivojski nevronski mreži ter $lambda1$ in $lambda2$ regularizacijska parametra v tro- in štirinivojski nevronski mreži. Funkcija zraven vseh matrik uteži, ki jih v nadaljevanju raziskovalne naloge ne bomo potrebovali, vrne še spremenljivki acc_3layer in acc_4layer, ki predstavljata natančnosti tri- in štirinivojske nevronske mreže.
Funkcija najprej nabor podatkov na podoben način kot funkcija *buildOneVsAll2Class()* razdeli na dve skupini. S podatki iz prve skupine izračuna optimalne vrednosti uteži s pomočjo funkcij *build1()* in *build2()*, nato pa s temi vrednostmi uteži izračuna pričakovano natančnost nevronske mreže. Postopek računanja pričakovane natančnosti je opisan v teoretičnem delu raziskovalne naloge, poglavje 4.7.

5.3.1 Izbira velikosti skritih nivojev brez regularizacije

Po implementaciji algoritma smo želeli preučiti vpliv velikosti skritih nivojev na pričakovano natančnost nevronske mreže. Za vsak nabor podatkov smo želeli izračunati natančnost za 5 različnih velikosti skritih nivojev. Prav tako smo zaradi faktorja naključja želeli za eno število skritih nivojev natančnost izračunati 5 krat (včasih zaradi kompleksnosti to ni bilo mogoče). Za enostavnejše delo smo napisali funkcijo *chooseArchitecture(database, hidden_layer_size, lambda, num_runs),* ki za nabor podatkov $database$, velikosti vseh skritih nivojev $hidden\_layer\_size$ in regularizacijski paramtere $lambda$ izračuna pričakovano natančnost num_runs krat. Funkcija vrne matriko vseh natančnosti $accs$, povprečno natančnost za trinivojsko nevronsko mrežo avg_acc1 in povprečno natančnost za štirinivojsko nevronsko mrežo avg_acc2. V tem poglavju smo za regularizacijski parameter vedno izbrali 0, saj smo tako močno pospešili hitrost računanja. Funkcijo smo torej klicali z *[accs, avg_acc1, avg_acc2] = chooseArchitecture(normalize(database), n, 0, 5),* kjer je $database$ baza podatkov (podobno kot v poglavju 5.2.1) in n število skritih nivojev.
Rezultati so predstavljeni v tabeli 2:

Tabela 2. Pričakovane natančnosti nevronskih mrež z različnim številom nevronov v skritih nivojih brez regularizacije (lasten arhiv).

Velikost skritih nivojev		1. nabor podatkov		2. nabor podatkov		3. nabor podatkov	
		1 sk. nivo	2 sk. nivoja	1 sk. nivo	2 sk. nivoja	1 sk. nivo	2 sk. nivoja
5	1.	1	1	0,972	1	0,965	0,974
	2.	0,967	0,967	1	0,972	0,982	0,982
	3.	0,967	1	0,972	0,972	0,965	0,956
	4.	0,967	1	1	1	0,965	0,991
	5.	1	1	0,889	0,944	0,965	0,945
	Povp.	0,98	**0,993**	0,967	0,978	0,968	0,97
10	1.	1	0,967	1	1	0,974	1
	2.	1	1	0,972	0,972	0,947	0,947
	3.	1	1	1	1	0,991	1
	4.	0,967	0,967	0,972	0,972	0,964	0,965

	5.	0,967	0,967	0,944	0,972	0,982	0,965
	Povp.	**0,987**	0,98	0,978	**0,983**	0,972	**0,975**
20	1.	0,933	0,967	1	0,972	0,974	0,965
	2.	1	1	0,972	0,972	0,965	0,965
	3.	1	0,967	0,972	0,944	0,991	0,982
	4.	0,967	0,933	1	1	0,982	0,974
	5.	0,967	1	0,972	1	0,974	0,974
	Povp.	0,973	0,973	**0,983**	0,978	**0,977**	0,972
30	1.	0,967	0,967	1	0,972	0,947	0,956
	2.	1	0,9	1	0,944	0,982	0,991
	3.	0,967	1	0,944	1	0,956	0,965
	4.	1	1	0,972	0,944	0,982	0,982
	5.	0,967	0,967	0,944	0,972	0,982	0,965
	Povp.	0,98	0,967	0,972	0,967	0,970	0,972
50	1.	0,933	0,97	0,972	0,972	/	/
	2.	0,967	1	0,972	0,972	/	/
	3.	1	0,933	0,972	0,917	/	/
	Povp.	0,967	0,967	0,972	0,954	/	/

S krepko pisavo so označene najboljše pričakovane natančnosti v posameznemu naboru podatkov. Opazimo, da je prvi nabor podatkov najbolj uspešen pri štirinivojski mreži s petimi nevroni in trinivojski mreži z 10 nevroni, medtem ko pa sta druga dva nabora podatkov najbolj uspešna pri štirinivojski mreži z 10 nevroni in trinivojski mreži z 20 nevroni. To lahko obrazložimo z tem, da je prvi nabor podatkov manj kompleksen in bolj linearno ločljiv kot druga dva.

Če povprečne natančnosti pri najbolj natančni arhitekturi (številu nivojev in nevronov) nevronske mreže primerjamo z natančnostjo linearne logistične regresije, ugotovimo, da se pri prvem naboru podatkov razlikujejo za **0,02,** pri drugem za **0,005** in pri tretjem za **0,006**. Te vrednosti so v okviru napake meritve, zato ne moremo trditi, da so nevronske mreže na naših naborih podatkov res boljši algoritmi kot linearna logistična regresija.

Zanemariti ne smemo tudi podatka najvišje natančnosti v posameznem naboru podatkov, ki je pri vseh naših naborih podatkov enak 1. Pri linearni logistični regresiji sta v prvem in drugem naboru podatkov najvišji natančnosti bili enaki, vendar je bila najvišja natančnost pri tretjem naboru enaka 0,991. **Zaradi tega lahko trdimo, da je na tretjem naboru podatkov smiselna uporaba algoritma nevronske mreže. Na prvem in drugem naboru podatkov uporaba nevronske mreže ni smiselna, saj je linearna logistična regresija imela enako najvišjo natančnost, vendar je bila hitrejša in bolj enostavna.**

5.3.2 Izbira regularizacijskega parametra

Po določitvi najboljšega števila nevronov v skritih nivojih smo želeli ugotoviti še, če uporaba regularizacije izboljša natančnosti. Želeli smo preizkusiti 5 različnih regularizacijskih parametrov na najboljši arhitekturi tri- in štirinivojske nevronske mreže za vsak nabor podatkov. Da bi si delo olajšali, smo napisali funkciji *chooseLambda1(database, hidden_layer_size, lambda_range)* in *chooseLambda2(database, hidden_layer_1_size, hidden_layer_2_size, lambda_range)*. Prva funkcija izračuna optimalne uteži za vse regularizacijske parametre shranjene v vektorju $lambda\_range$ v prvi skupini (deljenje na dve skupini poteka kot v prejšnjih funkcijah) baze podatkov $database$ in z številom nevrov v skritem nivoju trinivojske nevronske mreže $hidden\_layer\_size$. Za vse vrednosti optimalnih uteži izračuna pričakovano napako, ki je enaka $1 - natančnost$ (Ng, A., Week 6) v obeh skupinah, ter nariše graf v napake v odvisnosti regularizacijske paramtera. Krivuljo napake v prvi skupini riše z rdečo barvo, krivulj napake v drugi skupini pa z modro barvo. Funkcija *chooseLambda2()* naredi podobno v štirinivojski nevronski mreži. Funkciji smo klicali z naslednjo komando: *[err1, err2] = chooseLambda2(normalize(database), n1, n2, [0; 0.33; 0.66; 1; 1.2])* in *[err1, err2] = chooseLambda1(normalize(database), n1, [0; 0.33; 0.66; 1; 1.2])*, kjer so vrednosti $n1, n2$ števila nivojev pridobljena v prejšnem poglavju (vrednosti v tabeli označene krepko). Vse funkcije smo klicali dvakrat, saj smo bili omejeni z zmogljivostjo našega računalnika. Pomemben rezultat, ki ga dobimo, je napaka na drugi skupini. Tem manjša ko je napaka na drugi skupini, tem boljša je pričakovana natančnost na novih primerih (Ng, A., 2015, Week 6). Ker funkciji *chooseLambda1()* in *chooseLambda2()*

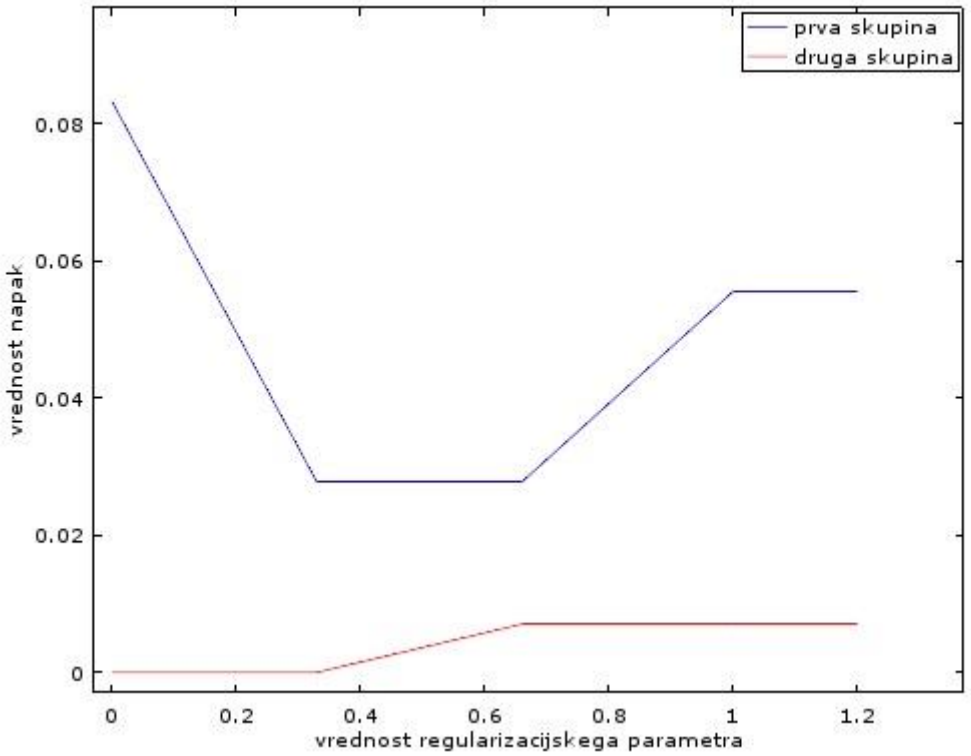
narišeta tudi graf, lahko iščemo minimum napake na drugi skupini (modre krivulje) tudi grafično.

Rezultati so zbrani v spodnji tabeli, v oklepajih pri številu skritih nivojev je zapisano število nevronov v njih:

Tabela 3. Napaka na drugi skupini nabora podatkov pri različnih vrednostih regularizacijskega parametra (lasten arhiv).

Vrednost regularizacijskega parametra		1. nabor podatkov		2. nabor podatkov		3. nabor podatkov	
		1 sk. nivo (10)	2 sk. nivoja (5)	1 sk. nivo (20)	2 sk. nivoja (10)	1 sk. nivo (20)	2 sk. nivoja (10)
0	1.	0	0,033	0,028	0,028	0,018	0,009
	2.	0,033	0	0	0,028	0,009	0,044
	Povp.	**0,017**	**0,017**	**0,014**	0,028	**0,014**	0,027
0,33	1.	0,033	0,033	0,028	0	0,035	0,009
	2.	0,067	0,033	0,028	0	0,009	0,035
	Povp.	0,05	0,033	0,028	**0**	0,022	**0,022**
0,66	1.	0,033	0,067	0,028	0	0,035	0,018
	2.	0,067	0,033	0,028	0,056	0,009	0,044
	Povp.	0,05	0,05	0,028	0,028	0,022	0,031
1	1.	0,067	0,067	0,056	0,028	0,044	0,018
	2.	0,067	0,067	0,028	0,056	0,018	0,035
	Povp.	0,067	0,067	0,042	0,042	0,031	0,027
1,2	1.	0,1	0,1	0,056	0,694	0,044	0,018
	2.	0,1	0,658	0,028	0,056	0,018	0,035
	Povp.	0,1	0,379	0,042	0,375	0,031	0,027

Graf 1. Primer grafa za izbiranje regularizacijskega parametra na drugem naboru podatkov (lasten arhiv).



Prikazujemo samo en primer grafa, ker za izbiro regularizacijskega parametra ni potreben, služi le za vizualizacijo.

Ugotovimo, da je regularizacija smiselna le pri štirinivojski nevronski mreži pri drugem in tretjem naboru podatkov, vendar tudi tam je primerna zelo majhna vrednost regularizacijskega parametra. Ponovno uporabimo funkcijo *chooseArchitecture()* za drugi in tretji nabor podatkov, regularizacijski parameter 0,33 in v poglavju 5.3.1 ugotovljeno arhitekturo ter si zapišemo pričakovane natančnosti pri štirinivojski nevronski mreži.

Tabela 4. Pričakovane natančnosti štirinivojskih nevronskih mrež pri drugem in tretjem naboru podatkov ter regularizacijskem parametru 0,33 (lasten arhiv).

	2. nabor podatkov	3. nabor podatkov
1.	1	0,974
2.	0,972	0,991

3.	0,972	0,982
4.	1	1
5.	0,972	0,965
Povprečje	0,983	0,982

Opazimo zelo podobno pričakovano natančnost kot pri enakih arhitekturah brez regularizacije. Pri drugem naboru z regularizacijo je povprečna pričakovana natančnost enaka kot brez regularizacije, pri tretjem naboru pa je nevronska mreža z regularizacijo natančnejša za 0,007. **Zaključimo lahko, da regularizacija na prvem in drugem naboru podatkov sploh ni smiselna. Na tretjem naboru podatkov lahko uporabimo majhno vrednost regularizacijskega parametra (0,33), ampak lahko pričakujemo le za manj kot odstotek večjo natančnost. Zaradi hitrejšega računanja je pri tretjem naboru podatkov tudi bolje uporabljati nevronsko mrežo brez regularizacije.**

5.4 Prikaz uporabe nevronskih mrež s pomočjo knjižnice Deeplearning4j

Za reševanje bolj kompleksnih problemov je smiselna uporaba različnih knjižnic, ki uporabljajo hitrejše programske jezike in bolj optimizirane algoritme. Za prikaz uporabe takšnih knjižnic smo v nalogi izbrali knjižnico Deeplearning4j, predvsem njen del Word2Vec, ki je napisana za programski jezik Java.

Algoritmi za pripravo vektorskega modela Word2Vec sprejmejo kot vhod besedilo in ga spremenijo v vektorski model, kjer je vsaki besedi prirejen vektor. Število dimenzij vektorskega modela je nastavljivo. Razdaljo med posameznimi vektorji je določena s kontekstom, v katerem se besede pojavljajo. Besede, ki imajo podoben kontekst, bodo imele pripadajoče vektorje bližje skupaj. Podobni besedni pari (npr. programer - programerka in voznik - voznica) bodo imeli podobne razdalje med pripadajočimi vektorji.
Za procesiranje teksta uporablja dvonivojsko nevronsko mrežo. Podpira dva načina učenja, pri prvem je vhod okolica besede, izhod pa beseda, pri drugem pa je vhod beseda in izhod okolica besede. Izkaže se, da je slednji način bolj primeren za velike količine podatkov (Deeplearning4j Development Team, 2016).

Za prikaz uporabe knjižnice Deeplearning4j smo uporabili besedilo dela angleške wikipedije, pridobljene iz spletne strani Wikipedia: Database Download (Wikipedia, 2015). S pomočjo mentorja smo besede postavili v nedoločnike (Manning C., Jurafsky, D. & Liang, P., 2015), odstranili pogoste besede brez konkretnega pomena in napisali program za kreiraje vektorskega modela. Vse datoteke povezane s programom so v mapi Deeplearning4j v prvi prilogi. V tem delu raziskovane naloge nismo nameravali opraviti meritev, temveč samo prikazati uporabo moderne knjižnice, zato programa ne bomo podrobneje opisovali. Vektorski model smo uporabili za predpostavljanje besed s podobnim kontekstom kot vhodna beseda. Nekaj rezultatov je predstavljenih v spodnji tabeli:

Tabela 5. Prikaz rezultatov vektorskega modela angleške wikipedije (lasten arhiv).

beseda	okolica besede		
photon	entropy	molecule	antiproton
entropy	thermodynamic	emission	electron
oscar	award	filmfare	best
computer	software	graphics	computing

Iz tabele je razvidno, da so rezultati precej dobri, saj so konteksti besed in predvidene okolice zelo podobni.

6. UGOTOVITVE IN ZAKLJUČEK

V raziskovalni nalogi smo se ukvarjali z uporabo algoritmov linearne logistične regresije in nevronskih mrež na treh različnih bazah podatkov. Baze podatkov smo pridobili iz spletne strani Machine Learning Repository (Lichman, M., 2013). Prva baza podatkov predstavlja klasifikacijo treh rož roda Iris (perunika), druga klasifikacijo treh različnih vrst italijanskega vina, tretja pa diagnosticiranje tumorjev na prsih.

Pri izbiri optimalne arhitekture nevronske mreže za vsak nabor podatkov smo ugotovili, da je pri prvem naboru podatkov najbolj natančna trinivojska nevronska mreža z **desetimi** nevroni v skritem nivoju ali štirinivojska nevronska mreža z **petimi** nevroni v vsakem skritem nivoju. Pri drugem in tretjem naboru podatkov je najbolj natančna trinivojska nevronska mreža z **dvajsetimi** nevroni v skritem nivoju ali štirinivojska nevronska mreža z **desetimi** nevroni v vsakem skritem nivoju. Iz teh ugotovitev lahko tudi sklepamo, da je prvi nabor podatkov najmanj kompleksen. Če preučimo pričakovane natančnosti pri drugem in tretjem naboru podatkov lahko tudi sklepamo, da je tretji nabor podatkov bolj kompleksen kot drugi, saj so pričakovane natančnosti pri slednjem manjše.

Iz poskusov za izbiro optimalnih vrednosti regularizacijskega parametra smo ugotovili, da je uporaba regularizacije smiselna le pri tretjem naboru podatkov z vrednostjo regularizacijskega parametra 0,33. Tudi v tem primeru je povprečna pričakovana natančnost večja le za 0,007 (0,7%). Za prvi in drugi nabor podatkov je enako ali bolj učinkovito uporabiti nevronsko mrežo brez regularizacije.

Pri poskusih z nevronskimi mrežami nas je rahlo omejevala tudi zmogljivost našega računalnika. Zaradi tega smo bili omejeni na uporabo le tri- in štirinivojskih nevronskih mrež z največ 50 nevroni v skritih nivojih in baz podatkov z manjšim številom lastnosti (največ 30) ter naborov lastnosti (največ 569). Ravno zaradi tega smo izmerili večje odstopanje pričakovanih natančnosti; pri linearni regresiji največ 3,5% in pri nevronskih mrežah največ 4%, razen pri velikih vrednostih regularizacijskega parametra, kjer so bile odstopanja tudi do 74%. Pri slednjih je bil razlog verjetno manj zmogljiv računalnik, ki ni uspešno izračunal optimalnih vrednosti uteži. Pri nevronski mreži je napaka večja zaradi nekonveksnosti

stroškovne funkcije. Funkcija za iskanje minimuma se zaradi tega lahko ujame v lokalnem minimumu.

V tabeli 6 smo zbrali povprečne pričakovane natančnosti in največje pričakovane natančnosti pri različnih algoritmih. Pod λ so zbrane vrednosti regularizacijskih parametrov, pod arhitekturo število nevronov v skritih nivojih, pod $\bar{a}$ povprečne pričakovane natančosti in pod a_{max} največje pričakovane natančnosti. Kratica LLR pomeni linearna logistična regresija, NM3 trinivojska nevronska mreža in NM4 štirinivojska nevronska mreža.

Tabela 6. Primerjava pričakovanih natančnosti vseh obravnavanih algoritmov (lasten arhiv).

	1. nabor podatkov			2. nabor podatkov			3. nabor podatkov		
Algoritem	LLR	NM3	NM4	LLR	NM3	NM4	LLR	NM3	NM4
λ	0	0	0	0	0	0	0	0	0,33
Arhitektura	/	10	5	/	20	10	/	20	10
$\bar{a}$	0,973	0,987	**0,993**	0,978	**0,983**	**0,983**	0,971	0,977	**0,982**
a_{max}	1	1	1	1	1	1	0,991	0,991	**1**

Iz zgornje tabele je razvidno, da je pri tretjem naboru podatkov najbolj smiselna uporaba štirinivojske nevronske mreže z vrednostjo regularizacijskega parametra 0,33, saj je pri tem algoritmu dosežena najvišja maksimalna pričakovana natančnost in najvišja povprečna pričakovana natančnost. Pri prvem in drugem naboru podatkov so najvišje pričakovane natančnosti pri vseh algoritmih enake, razlike v povprečnih natančnostih pa majhne. Izbiramo lahko torej med linearno logistično regresijo ali nevronsko mrežo brez regularizacije. Bolj smiselna je uporaba linearne logistične regresije, saj je hitrejša in zahteva manj zmogljiv računalnik.

Dobljeni rezultati (optimalni regularizacijski parameter, arhitekture, pričakovane natančnosti) veljajo samo za tiste nabore podatkov, na katerih smo jih dobili, in ne veljajo v splošnem. Za vsak nov nabor podatkov moramo s predstavljenimi postopki najprej izbrati optimalno arhitekturo nevronske mreže in optimalno vrednost regularizacijskega parametra, nato pa pričakovano natančnost primerjati z drugimi algoritmi.

Na koncu smo preizkusili še uporabo modernejše knjižnice Deeplearning4j. Ugotovili smo, da je njena uporaba relativno enostavna, rezultati pa dobri.

Veliko znanstvenikov se ukvarja z iskanjem novih algoritmov, ki bodo popolnoma spremenili strojno učenje. Algoritem vzvratnega razširjanja je trenutno eden izmed najmočnejših algoritmov na področju strojnega učenja, vendar se to lahko zelo hitro spremeni. Danes poznamo vsaj pet različnih algoritmov, ki so na določenem področju zelo dobri: algoritem vzvratnega razširjanja, evolucijski algoritem, algoritem Bayesovih mrež, induktivno logično programiranje ter algoritem analogij. Prehitel bi ga lahko na primer algoritem Bayesian mreže, ki posnema evolucijo, vendar še trenutno ni tako uspešen glede kot algoritem vzvratnega razširjanja (Dominogs, P., 2016). Iskanje posplošenega (master) algoritma se nadaljuje. Lahko, da bo to preplet naštetih algoritmov ali nekaj povsem novega.

6.1 Družbena odgovornost

Razvijanje algoritmov strojnega učenja je usmerjeno k izboljšanju kvalitete človekovega življenja in odnosa do narave. Moderni algoritmi pomagajo razkrivati različne goljufije, na primer sporna nakazila, odkrivanja povezav med spornimi dogodki itd. Prav tako so koristni v gospodarstvu in marketingu, saj samostojno odkrivajo želje in pričakovanja uporabnika ter napake v razvoju izdelkov ali v izvedbi storitev. Korist imajo tudi v ekološkem smislu, saj pomagajo iskati korelacije med naravnimi pojavi in napovedovati katastrofe, ki jih strokovnjaki lahko preprečijo.
Uporaba tehnologij strojnega učenja v raziskovalni nalogi ne dela razlike med različnimi spoli, starostmi, veri ali barvi kože ter spoštuje vse temeljne človekove pravice in norme obnašanja. Prav tako nima negativnega vpliva na okolje, prispeva pa k popularizaciji strojnega učenja.

7. SEZNAM VIROV IN LITERATURE

[1] Clark, L. (online). (Wired): DeepMind's AI is an Atari Gaming Pro Now. 25. 2. 2015 (citirano 19. 1. 2016). Dostopno na naslovu: http://www.wired.co.uk/news/archive/2015-02/25/google-deepmind-atari.

[2] Ng, A (online). (Coursera, Stanford University): Machine Learning. Session 30. 11. 2015 – 22. 2. 2016 (citirano 21. 1. 2016). Dostopno na naslovu: https://www.coursera.org/learn/machine-learning.

[3] Coursera (online): ML Wiki. 2016 (citirano 21. 1. 2016). Dostopno na naslovu: https://share.coursera.org/wiki/index.php/ML:Introduction.

[4] Lichman, M. (online). (Irvine, CA: University of California, School of Information and Computer Science): UCI Machine Learning Repository. 2013 (citirano 28.1. 2016). Dostopno na naslovu: http://archive.ics.uci.edu/ml.

[5] Deeplearning4j Development Team (online). (Deeplearning4j): Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0. 2016 (citirano 3. 2. 2016). Dostopno na naslovu: http://deeplearning4j.org/.

[6] Domingos, P. (online). (Wired): The race for the master algorithm has begun. 25. 1. 2016 (citirano 3. 2. 2016). Dostopno na naslovu: http://www.wired.co.uk/magazine/archive/2016/01/ideas-bank/master-algorithm-pedro-domingos.

[7] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. New York, Springer.

[8] Witten, I. H., Frank, E. & Hall, M. A. (2011). Data mining: practical machine learning tools and techniques. Amsterdam, Elsevier.

[9] Wikipedia (online). Wikipedia: Database Download. 11. 11. 2015 (citirano 3. 2. 2016). Dostopno na naslovu: https://en.wikipedia.org/wiki/Wikipedia:Database_download.

[10] Manning C., Jurafsky, D. & Liang, P. (online). Stanford NLP group. 2015 (citirano 3. 2. 2016). Dostopno na naslovu: http://nlp.stanford.edu/people/.

[11] Wikipedia (online). Linear Regression. 2. 2. 2016 (citirano 4. 2. 2016). Dostopno na naslovu: https://en.wikipedia.org/w/index.php?title=Linear_regression&action=history.

8. PRILOGE

1. DVD zgoščenka z vsemi opisanimi programskimi datotekami in bazami podatkov.