Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jan Pavlin

Globoko učenje na neslikovnih medicinskih podatkih

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2018

Copyright.

Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preglejte področje uporabe globokih nevronskih mrež na atributnih (neslikovnih) medicinskih podatkih. Implementirajte klasifikatorje z različnimi vrstami in topologijami globokih nevronskih mrež in izmerite njihove performanse. Posebno pozornost namenite delu z visokodimenzionalnimi podatki in številnimi razredi, ter obravnavi manjkajočih podatkov. Dobljene rezultate primerjajte z uveljavljeno metodo za atributno učenje xgboost, ter opredelite prednosti in slabosti nevronskih mrež v primerjavi z njo.

Zahvaljujem se mentorju dr. Matjažu Kukarju za strokovno pomoč, vodenje in nasvete pri izdelavi diplomske naloge. Posebno bi se rad zahvalil tudi svoji družini in punci Sari za spodbudo in podporo skozi celoten študij. Hvala.

Kazalo

Povzetek

Naslov: Globoko učenje na neslikovnih medicinskih podatkih

Avtor: Jan Pavlin

V zadnjem desetletju se vse bolj množično uporabljajo tehnike globokega učenja. Zaslugo za to lahko pripišemo razvoju tehnologije in zlasti znižanju cen grafičnih kartic, ki omogočajo hitro učenje. Globoko učenje je področje strojnega učenja in se je uveljavilo predvsem na področjih računalniškega vida, prepoznavanja govora, prepoznavanja slik, analize teksta. Uporaba tega področja počasi prodira tudi v medicino.

V diplomskem delu bom preučil področja globokega učenja, upravljanja s pomanjkljivimi podatki in upravljanja z neuravnoteženimi podatki. Zgradil bom modele različnih topologij globokih nevronskih mrež in med njimi primerjal dosežene rezultate na podatkovni množici medicinskih podatkov. Analiziral bom tudi uporabo grafične kartice in procesorja za učenje ter uporabo nevronskih mrež.

Cilj diplomske naloge je preizkusiti in analizirati uporabo globokega učenja na medicinskih podatkih z uporabo različnih pristopov k reševanju problema pomanjkljivih podatkov. Določiti je potrebno, katera topologija in katera metoda predprocesiranja podatkov se najbolje obneseta, kakšne rezultate dosežeta in koliko časa je potrebnega za učenje te nevronske mreže.

Ključne besede: globoko učenje, medicinski podatki, nevronske mreže.

Abstract

Title: Deep learning on medical data

Author: Jan Pavlin

Use of deep learning is increasing in the last decade. This is mostly because advancement of technology and also price reduction of various graphical processors, which allow quick learning. Deep learning is an area of machine learning and has been used for computer vision, speech recognition, image classification and other. Usage of deep learning is being slowly used in medicine.

In the diploma thesis, I will examine the areas of deep learning, management of insufficient data and management of unbalanced data. I will build models of different topologies of deep neural networks and compare the results achieved on the data set of medical data among them. I will also analyse the learning and use of a graphics card and a processor for learning and using neural networks.

The goal of the thesis is to test and analyse the use of deep learning on medical data using different approaches to solving the problem of poor data. It is necessary to specify which topology and which methods of data preprocessing are best managed, what results they achieve and how much time is needed to learn this neural network.

Keywords: deep learning, medical data, neural networks.

Poglavje 1

Uvod

Strojno učenje je področje umetne inteligence, osredotočeno na tiste oblike sistemov, ki se lahko sami učijo, odločajo in odkrivajo zakonitosti v podanih podatkih. Glavne vrste strojnega učenja so nadzorovano učenje, nenadzorovano učenje in spodbujevalno učenje. Del strojnega učenja predstavljajo nevronske mreže. [24, 30]

Nevronske mreže so programi, ki delujejo po vzoru človeških možganov in s pomočjo vnaprej označenih primerov skušajo razbrati zakonitosti v podatkih. Sestavljajo jih med seboj povezane procesne enote, imenovane nevroni. Nevronske mreže so v računalništvu dovolj uveljavljen koncept, a se je njihova uporaba razširila šele v zadnjih nekaj letih z boljšo tehnologijo in konvolucijskimi nevronskimi mrežami. Posebno področje strojnega učenja je globoko učenje. To obravnava večslojne nevronske mreže in se pojavlja predvsem na področjih računalniškega vida, prepoznave zvoka in procesiranja naravnega jezika. Modeli globokih nevronskih mrež so sestavljeni iz mnogih skritih slojev z namenom, da v podatkih abstrahirajo tudi bolj kompleksne relacije (vzorce). [30, 27]

1.1 Sorodna dela

V zadnjem desetletju se strojno učenje vse bolj pogosto uveljavlja v medicini. Najbolj uporabljene so konvolucijske nevronske mreže, ki se uporabljajo pri učenju na slikovnih podatkih. V raziskavi [12] so uporabili konvolucijske nevronske mreže za klasifikacijo kožnega raka, kjer je mreža dosegla boljšo točnost od zdravnikov. Podobno so v raziskavi [14] uporabili konvolucijske nevronske mreže pri klasifikaciji CT slik za bolezni pljuč in dosegli $87,9\%$ točnost.

Čeprav so konvolucijske nevronske mreže popularnejše, se pogosto uporabljajo tudi polnopovezane nevronske mreže, ki se uporabljajo na neslikovnih podatkih. Polnopovezane nevronske mreže so bile uporabljene pri diagnosticiranju in napovedovanju preživetja pacientov z rakom na debelem črevesju [6], za zgodnje diagnosticiranje diabetesa [33] in celo pri ugotavljanju ginekoloških bolezni [35]. Globoke nevronske mreže, ki se od polnopovezanih razlikujejo zgolj v številu skritih slojev, so bile v medicini uporabljene za povečano točnost in efektivnost v histopatoloških diagnozah [25] in za odkrivanje metastaznega raka dojk [36].

Uporabo globokih polnopovezanih nevronskih mrež na neslikovnih medicinskih podatkih pogosto otežujejo manjkajoči podatki, ki se v medicinskih podatkovnih množicah pojavljajo zelo pogosto. Razlogov za njihov nastanek je več, njihova prisotnost pa ovira učenje nevronskih mrež, ne da bi podatke ustrezno predprocesirali. [26, 19]

1.2 Motivacija in opis diplome

Za uporabo globokih nevronskih mrež smo se odločili zato, ker podatkovno množico sestavlja mnogo med seboj odvisnih rezultatov medicinskih preiskav. Globoke nevronske mreže so sposobne tudi večje abstrakcije relacij med podatki. Zanima nas, kako se izkažejo različne topologije in modeli globokih nevronskih mrež na podatkovni množici medicinskih podatkov z nepopolnimi atributi in neuravnoteženimi razredi. Preizkusili bomo

različne načine predprocesiranja podatkov z namenom, da modeli dosežejo večje točnosti ali boljše priklice posameznih razredov. Za reševanje problema manjkajočih podatkov bomo uporabili različne načine imputacij, za reševanje problema neuravnoteženih podatkov pa bomo izbirali med povečanjem oziroma zmanjšanjem števila podatkov ter hibridnimi metodami. Primerjali bomo čas, ki je potreben za učenje in uporabo nevronskih mrež na grafični kartici ali procesorju in komentirali prednosti ter slabosti drugih pristopov strojnega učenja v primerjavi z nevronskimi mrežami.

Diplomsko delo bo na začetku predstavilo teoretično ozadje o nevronskih mrežah in globokem učenju. Sledilo bo poglavje, imenovano podatki, kjer bo opisana podatkovna množica, s katero smo delali. Za njim bo še eno teoretično poglavje, v katerem bomo predstavili problem in rešitve manjkajočih ter neuravnovešenih podatkov. Sledil bo opis metod, s katerimi smo se naloge lotili, kako smo reševali probleme, na katere smo naleteli, in kakšne modele nevronskih mrež smo zgradili. Predzadnje bo poglavje rezultatov, ki jih bomo vizualno prikazali in komentirali. Na koncu bo sklep, kjer bodo predstavljene glavne ugotovitve v zvezi z zadanim problemom.

Poglavje 2

Nevronske mreže

Umetne nevronske mreže so skupek algoritmov, ki poskušajo identificirati relacije v podatkih. Njihovo delovanje posnema delovanje človeških možganov in živčnega sistema, pomagajo pa nam pri razvrščanju množic podatkov v razrede. [17]

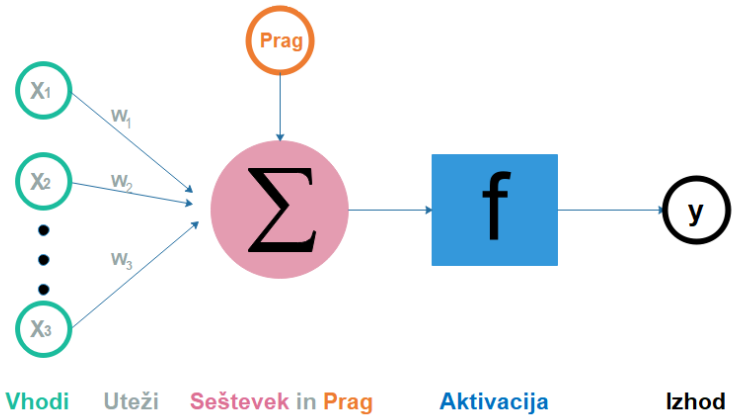
2.1 Nevron

Podobno kot pri človeških možganih je osnovni gradnik umetne nevronske mreže nevron. Nevroni so tipično povezani v sloje in ti so med seboj povezani. Smer povezave je ponavadi v smeri od vhodnega sloja proti izhodnemu. Nevron deluje tako, da prejme vhode iz zunanjih virov in za njih izračuna svojo izhodno vrednost. Sestavljen je iz vhodov, uteži, praga in aktivacijske funkcije. Matematični zapis za funkcijo nevrona je podan v enačbi 2.1

$$x_{out} = f(\sum_i w_i x_i + w_{bias}), \tag{2.1}$$

kjer so $x_1, x_2, x_3, ..$ vhodi nevrona, $w_1, w_2, w_3, ..$ pa njihove uteži. Funkcija f se imenuje aktivacijska funkcija, bias pa predstavlja prag. Vsak nevron torej svojo izhodno vrednost x_{out} dobi tako, da sprva sešteje vse z utežmi zmnožene vhodne vrednosti, nato rezultatu tega prišteje prag. Uteženo vsoto

kot argument posreduje aktivacijski funkciji, katere vrednost vrne nevron na izhodu. Zgradba nevrona je prikazana na sliki 2.1 [30, 17, 27]



Slika 2.1: Slika nevrona

2.2 Aktivacijska funkcija

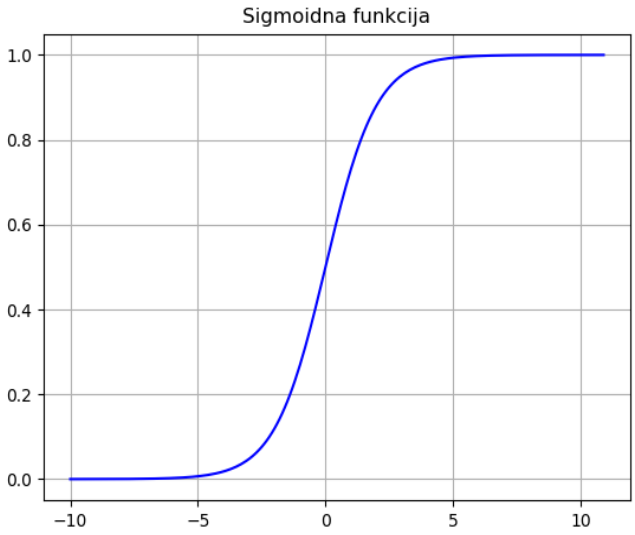
Aktivacijska funkcija se uporablja za preslikavo izhodne vrednosti nevrona na določen interval. Ta interval je najpogosteje med 0 in 1 ali med -1 in 1. Izbira aktivacijske funkcije ni predpisana, so pa nekatere bolj uporabne kot druge. V praksi so najbolj uporabljene sigmoidna funkcija, hiperbolični tangens in ReLU. [21, 5]

Sigmoidna funkcija

Sigmoidna ali tudi logistična aktivacijska funkcija slika vrednosti na interval med 0 in 1. Velika negativna števila preslika proti 0, velika pozitivna pa proti 1 (enačba 2.4). Ker je ločnica med obema razredoma dokaj strma, se funkcija pogosto uporablja za binarno klasifikacijo, saj tam želimo izračunati, ali vektor pripada nekemu razredu ali ne. Toda sigmoidna funkcija ima tudi pomanjkljivosti: funkcija je računsko zahtevna in ima problem izginjajočega

gradienta (angl. Vanishing gradients). Problem izginjajočega gradienta nastane ob robovih funkcije, kjer se njena vrednost približa vrednostnima 0 ali 1. Njen gradient je tam skoraj 0, učenje nevronske mreže pa je zaradi manjšega gradienta šibkejše. [21]

$$f(x) = \frac{1}{1 + e^{-cx}} \tag{2.2}$$

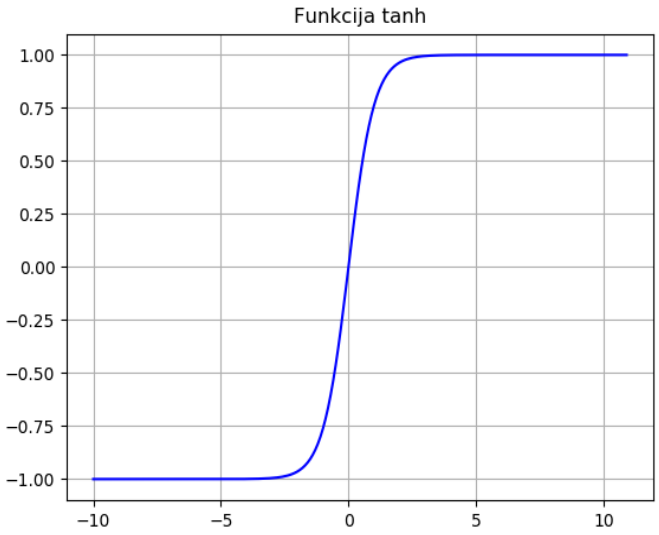


Slika 2.2: Graf sigmoidne funkcije

Hiperbolični tangens

Funkcija hiperbolični tangens ali krajše tanh je predelana sigmoidna funkcija. Iz enačbe 2.3 lahko ugotovimo, da slika vrednosti na interval med -1 in 1 ter ima center v točki 0. Funkcija se tako kot sigmoidna uporablja za binarno klasifikacijo, ima problem izginjajočega gradienta in je računsko zahtevna. [21]

$$f(x) = \frac{2}{1 + e^{(-2x)}} - 1 \tag{2.3}$$



Slika 2.3: Graf funkcije hiperboličnega tangensa

ReLU

Aktivacijska funkcija ReLU (angl. Rectified Linear Unit) vrne rezultat po enačbi 2.4.

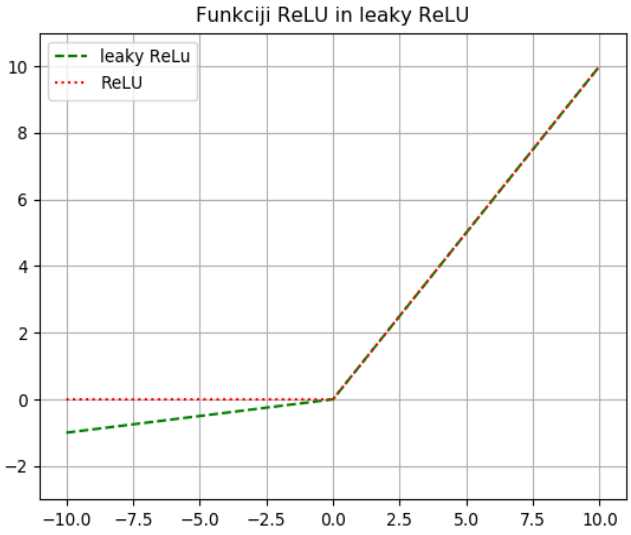
$$f(x) = max(0, x) \tag{2.4}$$

ReLU je najpogosteje uporabljena aktivacijska funkcija v skritih slojih nevronskih mrež. Funkcija za vhodne podatke, ki so manjši ali enaki nič, vrne rezultat nič, za ostale pa vrne njihovo vrednost. Rezultat funkcije je torej na intervalu med 0 in neskončno. Funkcija je računsko enostavnejša od sigmoidne funkcije ali hiperboličnega tangensa, saj je njena vrednost pri vseh negativnih vhodih enaka 0. Toda tudi funkcija ReLU ima slabost izginjajočega gradienta. Težava te funkcije je, da je pri vhodih, ki so manjši oz. enaki 0, gradient funkcije enak 0.

Temu problemu se lahko izognemo s funkcijo, ki se imenuje puščajoči ReLU (angl. Leaky ReLU). Edina razlika v njej je v tem, da namesto vre-

dnosti 0 pri $x < 0$ uporabimo dodaten parameter λ. Vrednosti λ se gibljejo na intervalu med 0 in 1, običajno pa imajo vrednost 0,1. Tako dobimo enačbo 2.5. Z uveljavitvijo dodatnega parametra λ gradient funkcije ni nikoli enak nič. [5]

$$f(x) = max(\lambda x, x); \lambda < 1 \tag{2.5}$$



Slika 2.4: Graf funkcij ReLU in leaky ReLU

2.3 Funkcija izgube

Napaka pri napovedi nevronske mreže se izračuna kot razlika med izhodom nevronske mreže in pravilnim izhodom. Funkcija, ki se uporablja za izračun te napake, se imenuje funkcija izgube (angl. loss function). Ob uporabi različnih funkcij izgube za enake napovedi bodo izračunani različni rezultati napake. Ena najbolj znanih funkcij izgube je koren povprečne kvadratne napake (angl. root mean square error) ali RMSE. V enačbi 2.6 y_i predstavlja pravilno vrednost izhoda za i-ti primer, $\hat{y}_i$ predstavlja napovedano vrednost izhoda za i-ti primer, n pa število vseh primerov.

$$Err = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{2.6}$$

V nevronskih mrežah se v veliki meri uporabljata predvsem funkciji binarna križna entropija in kategorična križna entropija. Prva se uporablja za računanje napak, ko imamo le dva razreda, druga pa, ko jih imamo več. Spodaj je navedena enačba kategorične križne entropije 2.7. [32, 27]

$$Err = -\sum_{i}(y_i * log(\hat{y}_i)) \tag{2.7}$$

Lastnost kategorične križne entropije je, da izračuna le napake pravilnih razredov (kjer je y_i enak 1), napake nepravilnih razredov pa ignorira. Ko so napovedane vrednosti pravilnih razredov nizke, so vrednosti logaritmov nizke, s tem pa so visoke napake funkcije izgube.

2.4 Optimizacijski algoritem

Bolj natančna kot je nevronska mreža, manjšo napako bo vrnila funkcija izgube. Funkcijo izgube želimo minimizirati z optimizacijskimi algoritmi.

Stohastični gradientni spust

Stohastični gradientni spust (SGD) je iterativna metoda, ki s pomočjo gradienta išče minimum funkcije. Zaradi svoje preprostosti in hitrosti je najširše uporabljen optimizacijski algoritem za učenje nevronskih mrež. Za določeno točko se iz parcialnih odvodov funkcije f izračuna gradientni vektor. Ta je usmerjen v smer največjega spusta funkcije. Gradientni vektor se izračuna po enačbi 2.8.

$$\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}) \tag{2.8}$$

Od začetne točke se algoritem nato premakne v smeri gradientnega vektorja. Premik iz ene točke v drugo se tako izračuna z enačbo 2.9.

$$x_{n+1} = x_n - \gamma \nabla f(x_n) \tag{2.9}$$

Parameter γ, s katerim utežimo velikost koraka, se imenuje hitrost učenja, njegova vrednost pa je običajno med 0,01 ter 0,1. Večja kot je hitrost učenja, večji korak v smeri vektorja naredimo. Postopek računanja gradientnega vektorja in premikanja v njegovi smeri algoritem ponavlja, dokler ne doseže lokalnega minimuma. [10]

ADAM

Stohastični gradientni sestop deluje zelo hitro, toda rezultati tega algoritma so veliko slabši od želenih. Glavna slabost je konstantna hitrost učenja, ker se algoritem premika enako hitro, neodvisno od oddaljenosti od lokalnega minimuma. Zato se uporabljajo drugi, boljši optimizacijski algoritmi, med katere spada tudi Adam (angl. Adaptive Moment Estimation). Prednost tega algoritma predstavlja uporaba momentov. Delujejo tako, da vsakemu premiku dodajo del vrednosti premika v prejšnjem koraku in s tem pomagajo, da se algoritem premika hitreje v pravilni smeri ter se zmanjšajo oscilacije v smeri stran od lokalnega minimuma. Adam izračuna dve vrsti momentov, to sta prvi in drugi moment. Prvi moment se izračuna po enačbi 2.10 in 2.11, drugi pa po enačbah 2.12 in 2.13.

$$m_w^{(t+1)} = \beta_1 m_w^{(t)} + (1 - \beta_1) g_t \tag{2.10}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1 - \beta_1^t} \tag{2.11}$$

$$v_w^{(t+1)} = \beta_2 v_w^{(t)} + (1 - \beta_2) g_t^2 \tag{2.12}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1 - \beta_2^t} \tag{2.13}$$

V navedenih enačbah je $m_w^{(t)}$ vrednost prvega momenta s pristranskostjo, $\hat{m_w}$ vrednost prvega momenta brez pristranskosti, $v_w^{(t+1)}$ vrednost drugega

momenta s pristranskostjo, $\hat{v}_w$ pa vrednost drugega momenta brez pristranskosti. Gradient v časovnem intervalu t predstavlja g_t. β_1 in β_2 predstavljata razpad prvega nivoja in razpad drugega nivoja momenta ter ju lahko spreminjamo. Običajna vrednost β_1 je 0,9, vrednost β_2 je 0,999. Ko sta momenta izračunana, se algoritem premakne v novo točko $w^{(t+1)}$ 2.14. Učni korak α se običajno nastavi na vrednost 0,001, vrednost ϵ pa na 10^{-8}. [22]

$$w^{(t+1)} = w^t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.14}$$

2.5 Algoritem vzvratnega širjenja napake

Nevronske mreže so lahko sestavljene iz enega ali več slojev nevronov. Prvi nivo je vhodni, število nevronov v tem nivoju je podano s številom vhodnih atributov. Število izhodnih nevronov v zadnjem, izhodnem nivoju je običajno enako številu razredov. Nevronsko mrežo s samo enim nivojem imenujemo enonivojska nevronska mreža in se uporablja za reševanje linearnih klasifikacijskih problemov. Nivoji, ki so med vhodnim in izhodnim nivojem, se imenujejo skriti nivoji. V praksi se večinoma uporablja med 2 in 10 skritih nivojev, lahko pa se jih uporablja tudi več.

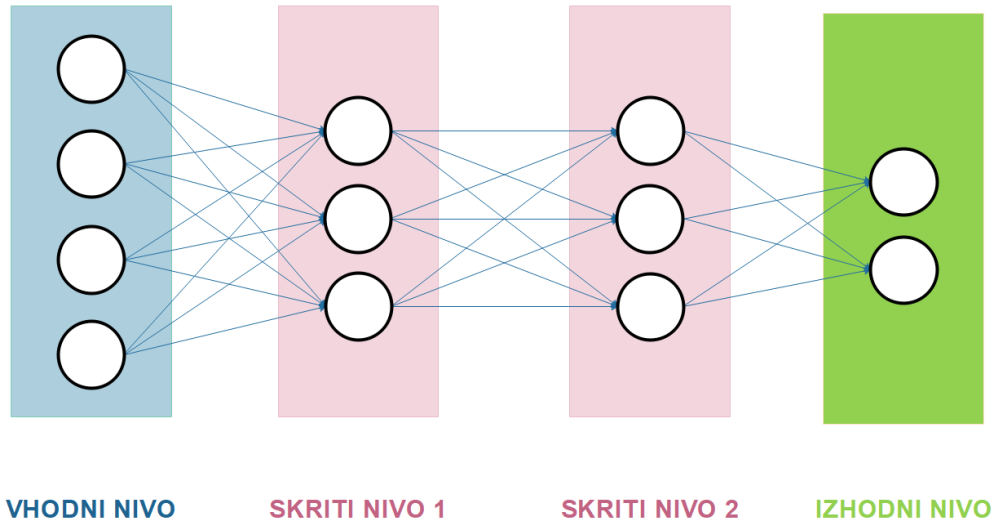
Za učenje nevronske mreže potrebujemo definiran model nevronske mreže z znanimi sloji, njihovimi povezavami, aktivacijskimi funkcijami, številom nevronov v vsakem sloju in začetnimi vrednostmi uteži. Pripravljena mora biti tudi množica vhodnih in izhodnih podatkov, na kateri se bo mreža učila. Nevronska mreža za vsak vektor vhodnih podatkov izračuna izhodno vrednost. Napaka računanja nevronov se dobi s primerjanjem izhodne vrednosti nevronske mreže s pričakovanim rezultatom. Algoritem, ki omogoča učenje nevronske mreže, se imenuje algoritem vzvratnega širjenja napake (ang. Backpropagation algorithm). Z uporabo tega algoritma mreža popravlja vrednosti uteži na povezavah med nevroni. Potrebna sprememba posamezne uteži se izračuna z optimizacijskim algoritmom, ki s funkcijo izgube izračuna napako posameznega nevrona. To se izvaja v obratnem vrstnem redu kot računanje izhodnih vrednosti v nevronski mreži. [17, 27]

2.6 Topologije nevronskih mrež

Glede na zgradbo in namembnost nevronskih mrež jih delimo na več tipov, za nas pa so pomembne le usmerjene nevronske mreže in konvolucijske nevronske mreže.

2.6.1 Usmerjene (feedforward) nevronske mreže

Usmerjene nevronske mreže sestavljajo trije osnovni sloji: vhodni sloj, n skritih slojev in izhodni sloj. Število skritih slojev je katerokoli nenegativno celo število. V usmerjenih nevronskih mrežah so povezave med sloji nevronov vedno povezane od vhodnega sloja v naslednjega proti izhodnemu. Pogosto se uporabljajo usmerjene nevronske mreže, kjer je vsak nevron i povezan z vsemi nevroni v naslednjem sloju. Te mreže se imenujejo polnopovezane usmerjene nevronske mreže. Primer polnopovezane usmerjene nevronske mreže je na sliki 2.5.



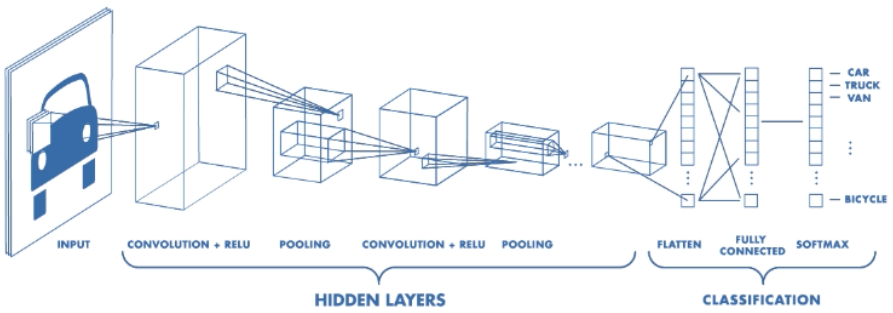
Slika 2.5: Nevronska mreža z dvema skritima slojema

Glede na število skritih slojev v usmerjenih nevronskih mrežah jih ločimo na dva tipa. To so globoke in plitve nevronske mreže. Globoke nevronske

mreže imajo mnogo skritih slojev za razliko od plitvih, ki imajo 1 ali 2 skrita sloja. Točne definicije o tem, koliko skritih slojev mora mreža imeti, da bo globoka, ni, ena izmed prvih globokih nevronskih mrež pa je imela 3 skrite sloje. Z uporabo globljih nevronskih mrež je možno identificirati kompleksnejše relacije v podatkih in s tem izboljšati njihovo točnost. [20, 18]

2.6.2 Konvolucijske nevronske mreže

Pri problemu klasifikacije slik in prepoznavanju vzorcev v njih so bile preproste nevronske mreže manj učinkovite, zato so se razvile konvolucijske nevronske mreže (CNN). Ta tip nevronskih mrež je drugačen od ostalih, poleg slojev navadnih nevronskih mrež ga namreč sestavljajo še konvolucijski sloji, sloji združevanja in sloji sploščitve. Na sliki 2.6 je predstavljen primer delovanja CNN. [28]



Slika 2.6: Konvolucijska nevronska mreža, vir:[3]

Konvolucijski sloji

Konvolucijski sloji so osrednji sloji konvolucijskih nevronskih mrež. Podatki, ki pridejo do njih, gredo skozi drseče sloje oziroma filtre, kjer se izluščijo pomembne lastnosti, kot so vzorci, robovi ali barve. Filtri so matrike, manjših dimenzij od vhodnih podatkov, in se pomikajo po vseh koordinatah vhodnega

prostora. Med pomikanjem filtrov skozi vhodne podatke se za vsako vrednost izračunajo vsote elementov, pomnoženih z utežmi filtra. [28]

V posameznem konvolucijskem sloju gredo lahko vhodni podatki skozi več filtrov, s tem pa se razberejo različne lastnosti podatkov. Če imamo vhodni prostor dimenzij $N \times N$, na katerem uporabimo filter w velikosti $m \times m$, bo dimenzija izhodnega sloja $(N - m + 1) \times (N - m + 1)$. Za vsako izmed vrednosti x_{ij}^l izračunamo novo po enačbi 2.15

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{ab} y_{(i+a)(j+b)}^{l-1} \tag{2.15}$$

Konvolucijski sloji lahko delujejo na eno-, dvo- ali trodimenzionalnih vhodnih podatkih. Pri enodimenzionalnih vhodnih podatkih se uporabljajo enodimenzionalni filtri, ki se pomikajo po vhodnem vektorju.

Sloji združevanja

Cilj uporabe slojev združevanja je zmanjšanje dimenzij vhodnih podatkov in s tem tudi zmanjšanje računske ter časovne zahtevnosti modela. Sloji združevanja tako podatke zgostijo in zmanjšajo, poskušajo pa ohraniti pomembne lastnosti vhodnih podatkov. Običajno se v konvolucijskih nevronskih mrežah uporabljajo sloji maksimalnega združevanja, velikosti 2×2. Tej zmanjšajo dimenzijo podatkov na 25% vrednosti, v vsakem oknu podane velikosti pa se ohrani najvišja (maksimalna) vrednost. V praksi se med vsakih nekaj konvolucijskih slojev pojavi sloj združevanja. [28]

Sloji združevanja so pri enodimenzionalnih podatkih velikosti N, pri dvodimenzionalnih $N \times M$ in pri trodimenzionalnih $N \times M \times O$.

Sploščitev

S pomočjo konvolucijskih slojev in slojev združevanja se iz vhodnih podatkov pridobijo oblike, vzorci in ostale lastnosti. Sploščitev je zaključni korak, preden gredo podatki iz konvolucijskega dela v polnopovezano mrežo. Sloj

sploščitve spremeni dimenzije vhodnih podatkov tako, da jih lahko kot vhodne sprejme polnopovezan sloj.

Poglavje 3

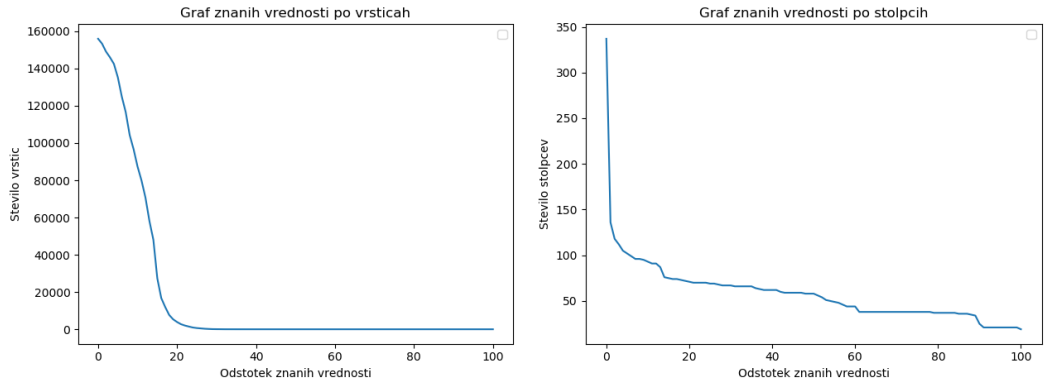
Podatki

Za analizo smo dobili podatke medicinskih preiskav in postavljenih diagnoz v datoteki csv (angl. comma seperated values). Podatkovna množica vsebuje 157096 vrstic (vektorjev) in 337 stolpcev (atributov). Vrstice predstavljajo posamezne preiskave, stolpci pa tipe preiskav. Vsi podatki so normalizirani in anonimizirani. Normalizirani so na interval $[-1, 1]$, torej so realne vrednosti preiskav preslikane tako, da ni nobena vrednost manjša od -1 ali večja od 1. To, da so podatki anonimizirani, pomeni, da ne vsebujejo nobenih podatkov o osebah, za katere so bile preiskave narejene, hkrati pa so šifrirana tudi imena tipov preiskav in diagnoz (razredov). V predzadnjem stolpcu so navedene anonimizirane šifre posameznih oddelkov, kjer je bil pacient obravnavan, v zadnjem stolpcu pa so navedeni tipi postavljenih diagnoz oziroma razredi. Šifre oddelkov smo uporabili le pri imputaciji manjkajočih vrednosti z metodo k-najbližjih sosedov. Vseh razredov je 353, pri čemer so njihove zastopanosti zelo neenakomerno porazdeljene.

Največji problem za klasifikacijo podatkov brez dvoma predstavljajo manjkajoče vrednosti, saj je le 11.6% vrednosti v množici atributov znanih. Ker smo menili, da je napovedna točnost najbolj odvisna od uspešnosti razrešitve tega problema, smo upravljanju z manjkajočimi podatki namenili največ pozornosti. Imputacijo smo izvedli na več različnih načinov in jih med seboj primerjali.

Grafa 3.1a in 3.1b prikazujeta deleže znanih podatkov v primerov in atributov. Iz prvega lahko razberemo, da imajo skoraj vsi primeri do največ 20% znanih vrednosti, medtem ko primerov z več kot 30% znanih vrednosti praktično ni. Krivulja odstotka znanih vrednosti po atributih je drugačna. To pa predvsem zato, ker obstajajo nekateri atributi, ki imajo skoraj vse vrednosti znane, dve tretjini atributov pa imata manj kot 10% znanih vrednosti.

Matrika znanih vrednosti je na sliki 3.2 in je velikosti 157096×337. Znotraj matrike je vsaka znana vrednost prikazana s črno barvo, vsaka neznana pa z belo. Matrika podatkovne množice, ki bi imela vse vrednosti znane, bi bila v celoti črna. V našem primeru so jasno črno obarvani le nekateri stolpci, ki imajo znanih veliko vrednosti.

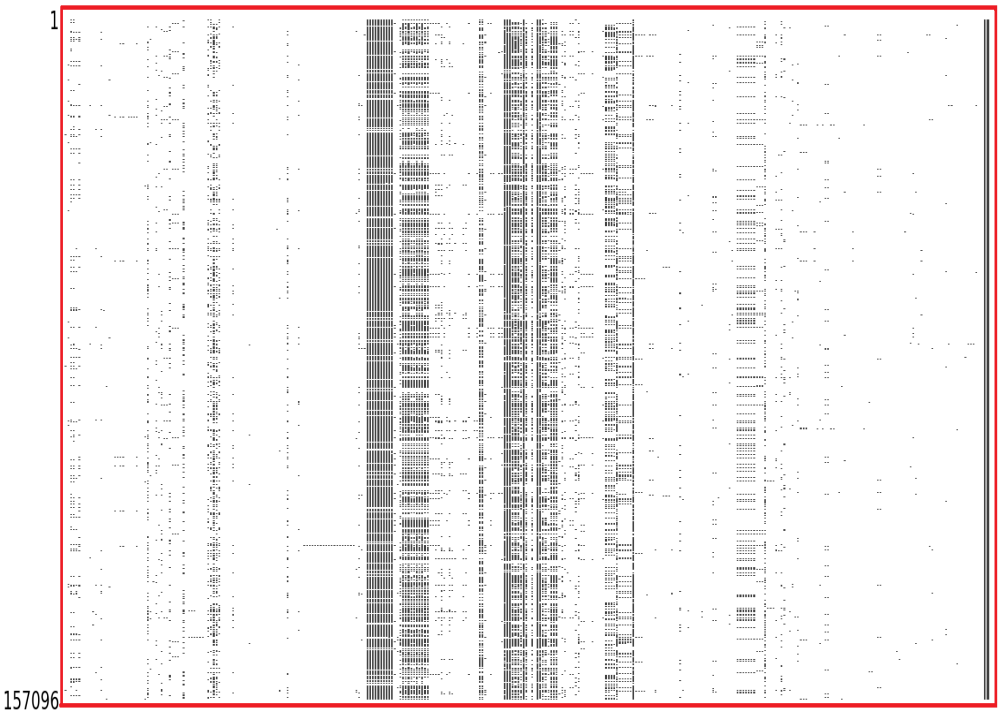


(a) Delež znanih podatkov po primerih (vrsticah)

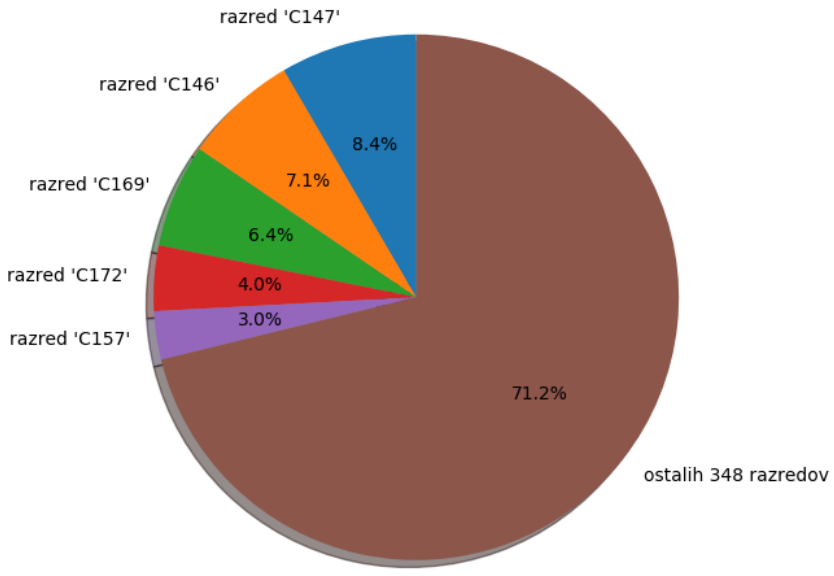
(b) Delež znanih podatkov po atributih (stolpcih)

Slika 3.1: Deleža znanih podatkov po vrsticah in stolpcih

Dodaten izziv predstavljajo neuravnoteženi podatki. Med 353 razredi jih je 5 najbolj zastopanih v skoraj 30% primerov. (3.3) Z imputacijo, ki se je izkazala za najboljšo, smo poskusili še dodatne metode, kako se soočiti z neuravnoteženimi podatki.



Slika 3.2: Matrika znanih vrednosti



Slika 3.3: Pogostost razredov v podatkovni množici

Poglavje 4

Upravljanje s podatki

4.1 Problem manjkajočih podatkov

Pri analizi in pripravi podatkov je zelo pomembna prisotnost manjkajočih podatkov. Razlogov za manjkajoče podatke v medicini je mnogo, na primer smrt pacienta, napake v merilnih napravah, neodgovarjanje na nekatera vprašanja ipd. Podatkovne množice z manjkajočimi podatki predstavljajo veliko težavo pri strojnem učenju in pri podatkovnem rudarjenju. Navkljub pogostim podatkovnim množicam z manjkajočimi podatki skoraj vsi algoritmi za strojno učenje pričakujejo polne podatke in na manjkajoče podatkovne množice niso prilagojeni. To je zato, ker pogosto izvajajo matrične ali podobne operacije na grafičnih karticah, ki ne sprejemajo praznih vrednosti.

Raziskovalci so tako prisiljeni, da se spoprimejo z manjkajočimi podatki z brisanjem, menjanjem ali imputacijo podatkov.

4.1.1 Tipi manjkajočih podatkov

Manjkajoče podatke delimo na tri skupine. To so podatki, ki manjkajo popolnoma naključno, podatki, ki manjkajo naključno in nenaključno manjkajoči podatki.

Prva skupina so podatki, ki manjkajo popolnoma naključno (angl. Missing completly at random) ali MCAR. Podatki manjkajo popolnoma na-

ključno takrat, ko je verjetnost, da bo nek podatek manjkajoč, neodvisna od vseh znanih in neznanih vrednosti, kot tudi od vrednosti podatka samega (enačba 4.1). V teh podatkih torej vrednosti manjkajo naključno, brez prepoznavnih vzorcev. V tej skupini manjkajočih podatkov se lahko uporabijo poljubne metode upravljanja s podatki, brez tveganja, da se bo kvaliteta podatkov močno poslabšala.

$$P(Y_{miss}|Y,X) = P(Y_{miss}) \Rightarrow Y_{miss} \perp\!\!\!\perp Y,X \tag{4.1}$$

Druga skupina so podatki, ki manjkajo naključno (angl. Missing at random) ali MAR. V tej skupini je verjetnost, da bo podatek neznan, odvisna od znanih vrednosti, ampak ne od vrednosti podatka samega (enačba 4.2).

$$P(Y_{miss}|Y,X) = P(Y_{miss}|X) \Rightarrow Y_{miss} \perp\!\!\!\perp Y \tag{4.2}$$

Zadnja skupina manjkajočih podatkov so nenaključno manjkajoči podatki (angl. Not missing at random). Če na verjetnost, da bo podatek manjkal, vpliva vrednost podatka samega, spada ta podatek med nenaključno manjkajoče. Delo s tem tipom manjkajočih podatkov je zahtevno, saj lahko vsaka sprememba podatkov poslabša njihovo kvaliteto. Tega tipa so tudi naši podatki, saj na to, ali bo podatek manjkal, vpliva njegova lastna vrednost. [15, 16]

$$P(Y_{miss}|Y,X) = P(Y_{miss}|Y,X) \Rightarrow Y_{miss} \not\perp\!\!\!\perp Y,X \tag{4.3}$$

4.1.2 Upravljanje z manjkajočimi podatki

V preteklosti je bilo razvitih več različnih načinov, kako se soočiti z manjkajočimi podatkovnimi množicami. Novejši, boljši pristopi sčasoma zamenjajo starejše, a ta proces se odvija zelo počasi, zato se še vedno pogosto uporabljajo starejše, slabše metode. Na to, kako se bo upravljalo z manjkajočimi podatki, vpliva veliko dejavnikov: koliko podatkov imamo, kolikšen del podatkov je manjkajoč, kakšnega tipa so manjkajoči podatki,... Opisanih je več načinov upravljanja z manjkajočimi podatki.

- Ignoriranje in brisanje podatkov: Obstajata dve glavni metodi, kako odstraniti podatke z manjkajočimi vrednostmi. Prva se imenuje analiza popolnih enot (angl. Complete case analysis) in se uporablja v večini primerov. Metoda deluje tako, da odstrani vse vektorje, ki vsebujejo vsaj en podatek neznane vrednosti. Glavna prednost te metode je njena preprostost, njena slabost pa je izguba informacij. Ta metoda se zato uporablja na podatkovnih množicah, ki imajo velik delež vektorjev z vsemi znanimi vrednostmi.[16, 15]

- Druga metoda, angl. discarding instances and/or attributes, deluje nekoliko drugače. Pri tej metodi se prepozna vektorje ali atribute (stolpce), ki imajo veliko manjkajočih vrednosti. Nato se oceni, kako pomembni so z vidika analize. Nepomembne atribute in vektorje se zavrže, pomembne pa se, tudi če imajo veliko manjkajočih podatkov, obdrži. Ignoriranje in brisanje podatkov je mogoče le, ko so manjkajoči podatki tipa MCAR. Te metode se ne uporablja pri podatkovnih množicah, ki imajo velik delež manjkajočih podatkov, saj bi zaradi tega izgubili preveč informacij. [15]

- Dodajanje dodatnih spremenljivk (angl. dummy variable adjustment). Manjkajoče podatke pridobimo z enim izmed načinov imputacije, k vsakemu podatku pa dodamo dodatno spremenljivko, ki označuje, ali je bil podatek znan ali ne. Podoben pristop temu načinu je tudi vstavljanje konstantnih vrednosti namesto manjkajočih. Le-te so lahko znotraj ali izven intervala normiranih vrednosti. Navkljub pogosti uporabi teh algoritmov je dokazano, da poslabšajo kvaliteto podatkov. [7]

- Imputacija - je skupina vseh algoritmov, ki zamenjajo nepoznane vrednosti z ocenami. Cilj teh algoritmov je posnemati razmerja med znanimi podatki in tako določiti vrednosti manjkajočih, da bi čim manj pokvarili kvaliteto podatkov. Najbolj osnovni načini imputacije so vstavljanje povprečne ali najbolj pogoste vrednosti (angl. mean substitution, median substitution).

- Preprosta imputacija. Je najbolj enostaven način imputacije. Manjkajoči podatek se zamenja z enim od znanih podatkov. Pogosto se uporabi kar zadnji znan podatek (angl. last value carried forward). Takšen tip imputacije je zelo površen, saj podatke vstavimo neodvisno od znanih podatkov v vektorju.

- Vstavljanje matematičnih sredin namesto manjkajočih vrednosti vstavi mediano ali povprečno vrednost vseh znanih vrednosti tega atributa. Ta metoda se uporablja zelo pogosto, saj je hitra in enostavna rešitev, vendar pa je pogosto tudi neprimerna. Kot pri preprosti imputaciji je tudi njena slabost, da je vstavljena vrednost neodvisna od vseh ostalih znanih vrednosti v vektorju, poleg tega pa se pri vstavljanju vrednosti gostijo okoli aritmetične sredine ali mediane. Metoda zaradi tega spada med slabše, njena uporaba pa je smiselna le, ko so manjkajoči podatki tipa MCAR. [15]

- Metoda k-najbližjih sosedov. Ta metoda izbere vektorju najbližje sosede na podlagi računanja medsebojnih razdalj. Najpogosteje se uporablja Evklidska razdalja, ki se izračuna po spodnji enačbi:

$$d(A, B) = d(B, A) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - y_i)^2}{n}}, \tag{4.4}$$

  kjer x_i predstavlja i-ti element vektorja A, y_i pa i-ti element vektorja B. Razdalja med vektorjema se računa le za elemente, ki so znani pri obeh vektorjih; da pa se dobi normirana vrednost, se razdalja deli s številom teh elementov (n). V primeru, da sta oba elementa vektorjev A in B neznana ali pa je neznan vsaj eden izmed njiju, se ta elementa in njuno razdaljo ignorira.

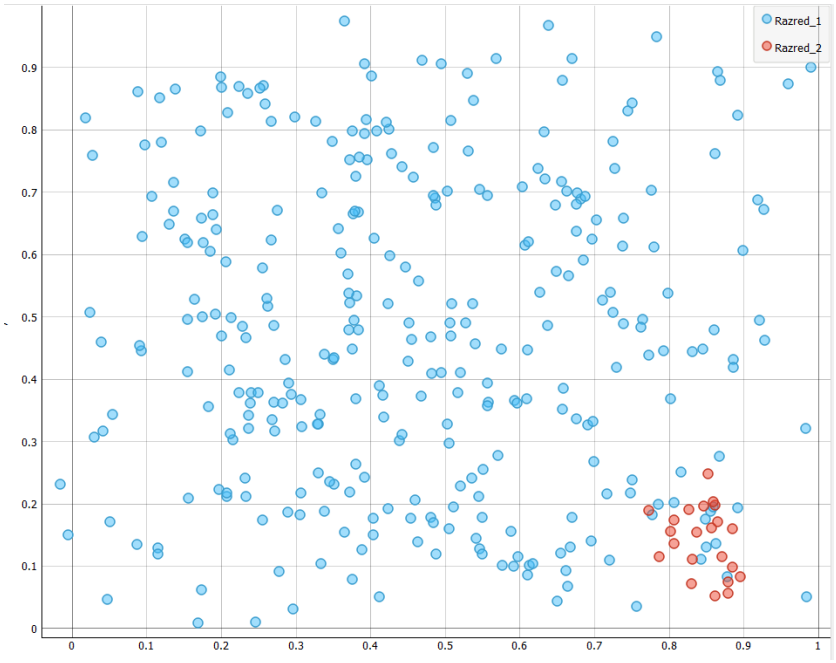
  Uporabljajo se tudi Manhattanska, Jaccardova in druge razdalje. Bolj kot sta si vektorja podobna, manjše so njune medsebojne razdalje. Ko se za določen vektor izračunajo vse njegove razdalje z ostalimi, se določi k sosedov, s katerimi ima čim manjšo medsebojno razdaljo. Nato se

izračuna povprečna vrednost spremenljivk k-tih najbližjih sosedov. [15, 7, 9]

- MICE (Multivatiate imputation by chained equations) Metoda MICE predpostavlja, da so manjkajoče vrednosti naključno manjkajoče (MAR), kar pomeni, da je verjetnost, da bo podatek manjkajoč, odvisna zgolj od znanih vrednosti. Metoda sprva s preprosto imputacijo ali vstavljanjem matematičnih sredin manjkajočim podatkom dodeli začasne vrednosti. Nato se vsako od začasnih vrednosti pretvori nazaj v manjkajočo. Manjkajoče vrednosti ponovno izračuna glede na porazdelitve ostalih vrednosti v trenutnem atributu. Tako vse manjkajoče podatke ponovno izračuna in jim zapiše vrednosti. Celoten zgornji postopek predstavlja eno iteracijo delovanja metode MICE. Na koncu vsake operacije so manjkajoče vrednosti zamenjane s predvidenimi, ki predstavljajo odnose v podatkih. Iteracije se lahko večkrat ponovijo, pri čemer so manjkajoče vrednosti posodobljene po vsaki iteraciji. Večinoma se v raziskavah uporablja deset iteracij, čeprav je optimalno število za vsako podatkovno množico različno. [8]

4.2 Neuravnoteženost podatkov

Če želimo da se algoritem strojnega učenja prilagodi vsem razredom enako, se morajo v podatkovni množici približno enako pogosto pojavljati. Toda v praksi imajo podatkovne množice neenakomerno porazdeljene razrede. Zelo jasen primer tega so medicinski podatki, kjer so redke bolezni manj zastopane v raziskavah kot druge. Podatki, kjer so nekateri razredi bolj zastopani kot drugi predstavljajo izzive pri strojnem učenju, saj se algoritmi bolj prilagodijo večinskim razredom. Prav tako so po navadi manjšinski razredi bolj pomembni za podatkovno rudarjenje, saj zaradi svoje redkosti nosijo pomembne in uporabne informacije. [23] Na sliki 4.1 je prikazan nabor neuravnoteženih podatkov. Ker je prvi razred bolj pogost, se bo algoritem strojnega učenja bolj prilagodil njemu.



Slika 4.1: Neuravnoteženi podatki

Pristope k učenju iz neuravnoteženih podatkov bi lahko ločili na tri glavne sklope:

- Metode na nivoju podatkov (angl. Data-level methods), ki spremenijo množico podatkov, tako da so razredi enakomerno razporejeni za strojno učenje. Na ta način pristopi dodajajo in brišejo podatke. Najbolj uporaben način je generiranje (podvojevanje) novih primerov manjšinskih razredov (angl. oversampling). Nasprotno od njega deluje zmanjševanje števila podatkov (angl. Undersampling) pristop, ki briše primere večinskih razredov.

- Metode na nivoju algoritmov (angl. Algorithm-level methods), ki direktno vplivajo na algoritem strojnega učenja, da le-ta posveti več pozornosti manjšinskim razredom in zanemari večinske. Za uporabo teh metod je potrebno poglobljeno znanje delovanja algoritmov strojnega učenja.

- Hibridne metode (angl. Hybrid methods), ki združujejo prednosti zgornjih dveh algoritmov. Ker je razvoj takih metod zahteven, se zelo redko uporabljajo.

4.2.1 Metode na nivoju podatkov

Od vseh treh zgoraj navedenih načinov, je najlažji način spreminjanja podatkovne možice (Data-level methods).

Zmanjševanje števila podatkov

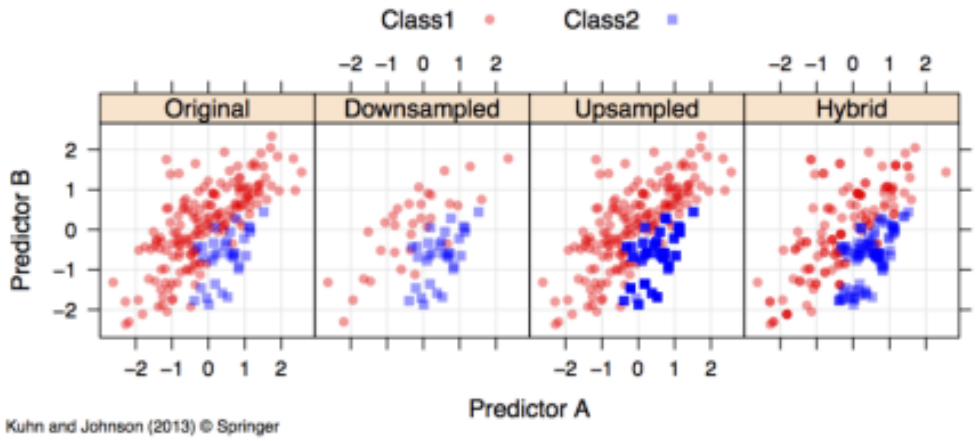
Zmanjševanje podatkovne množice (angl. undersampling) je način, ki zmanjša število instanc večinskih razredov do te mere, da so vsi razredi enako zastopani.

Povečevanje števila podatkov

Povečevanje podatkovne množice (angl.oversampling) deluje tako, da podvaja instance manjšinskih razredov, dokler niso razredi enako zastopani. Ker ta pristop ne briše podatkov in s tem ne izgubi nobene informacije, se pogosteje uporablja.

Hibridna metoda

Hibridna metoda uporabi tako povečevanje števila instanc manjšinskih razredov, kot tudi zmanjševanje instanc števila večinskih razredov. Na ta način ostane število instanc v podatkovni množici približno enako. Na sliki 4.2 je prikazano delovanje vseh treh metod nad neuravnoteženimi podatki. [34]



Slika 4.2: Neuravnoteženi podatki

Poglavje 5

Uporabljene metode

5.1 Predprocesiranje

Ko smo podatke prebrali, smo jih ločili na dva dela. Prvi del je vseboval atribute, drugi pa razrede. Nato smo razrede spremenili s kodiranjem ena naenkrat (angl. one hot encoding). To kodiranje za vsako vrednost priredi vektor, čigar dolžina je enaka številu vseh različnih razredov. Vektor ima pozitivno le tisto vrednost, ki predstavlja 'indeks' razreda, ki jo je imel primer, vse ostale vrednosti so nastavljene na 0.

Atribute smo predprocesirali na več načinov. Med načine predprocesiranja spada tudi normaliziranje vrednosti atributov na določen interval, brisanje praznih atributov ali brisanje atributov, pri katerih so vse vrednosti enake. Normaliziranja nismo uporabili, saj so podatki že bili na intervalu med -1 in 1, prazne atribute in atribute, kjer so vse vrednosti enake, pa smo izbrisali. Atributa, kjer so bile vse vrednosti manjkajoče, sta bila dva, atributi, kjer so bile vse vrednosti enake, pa so bili štirje.

5.2 Obravnava manjkajočih podatkov

- Vstavljanje konstantnih vrednosti: Namesto manjkajoče vrednosti smo vstavili konstantno vrednost 0.

- Vstavljanje povprečnih vrednosti: Namesto manjkajoče vrednosti smo vstavili povprečje vseh znanih vrednosti določenega atributa.
- Vstavljanje mediane: Namesto manjkajoče vrednosti smo vstavili vrednosti mediane posameznih atributov.
- Metoda k-najbližjih sosedov: Podatkovno množico smo ločili glede na šifre oddelkov, kjer so raziskave potekale, nato pa smo po skupinah za vsako vrstico našli k-najbližjih vrstic. Namen ločevanja po oddelkih je bil zmanjšanje časovne in prostorske zahtevnosti te metode. Vrednost k smo nastavili na 10. Neznane podatke v vrsticah vrstice smo imputirali z vstavljanjem povprečja znanih podatkov najbližjih vrstic. Če je določen podatek manjkal v vseh vrsticah najbližjih sosedov, smo vstavili konstantno vrednost 0.
- Metoda MICE: Manjkajoče vrednosti smo imputirali s pomočjo algoritma MICE. Uporabili smo 30 iteracij, kar je na naših podatkih trajalo 8 ur. Število iteracij bi bilo lahko večje, a smo ga omejili zaradi časovne zahtevnosti.

Testna metodologija, ki se je obnesla najbolje, je bila uporabljena tudi za preverjanje pristopov razreševanja neuravnoteženih podatkov. Napovedno točnost smo preverili s 5-kratnimm prečnim preverjanjem. Ta razdeli podatke na 5 enako velikih delov, nato pa enega od delov določi za testiranje. Na ostalih učnih množicah podatkov se nevronske mreže učijo, na testni pa se preveri klasifikacijska točnost. Postopek se izvede K-krat, dokler niso bili vsi deli tudi testne množice. Ker smo uporabljali poravnano prečno preverjanje (angl. Stratified K-fold), so se tudi razredi približno enako porazdelili, kot so porazdeljeni v celotni množici podatkov.

5.3 Izgradnja modela

Za izgradnjo in delo z nevronskimi mrežami smo uporabili orodje Keras [11]. Keras je orodje napisano v Pythonu, ki ponuja enostavno delo z različnimi

tipi nevronskih mrež. V zaledju lahko izbiramo med knjižnicami TensorFlow [4], CNTK [31] ali Theano. Z orodjem Keras lahko nevronske mreže učimo in uporabljamo s pomočjo CPU-ja ali GPU-ja. Izbrali smo knjižnico TensorFlow, ki jo je razvila ekipa Google Brain [4]. Zgradili smo globoke nevronske mreže in konvolucijske nevronske mreže, ki so bile sestavljene iz različnih slojev.

```
# izgradnja modela:
model = Sequential()

# dodaj vhodni sloj
model.add(Dense(num_neurons_per_layer, input_shape=(x_train.shape[1],)))
model.add(LeakyReLU())

# dodaj n-skritih slojev
for _ in range(num_hidden_layers):
    model.add(Dense(num_neurons_per_layer))
    model.add(LeakyReLU())
    model.add(Dropout(0.25, noise_shape=None, seed=None))

# dodaj izhodni sloj z aktivacijo
model.add(Dense(y_train.shape[1], activation='sigmoid'))

# zgradi model
model.compile(loss=keras.losses.categorical_crossentropy, # funkcija izgube
                  optimizer=keras.optimizers.Adam(), # optimizacijski algoritem
                  metrics=['accuracy'])

# nauči model na učni množici
history = model.fit(x_train, y_train,
                batch_size=batch_size,
                epochs=epochs,
                verbose=0,
                validation_data=(x_test, y_test),
                callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=5,
                                                         verbose=0, mode='auto', baseline=None)]
                    )
```

Slika 5.1: Koda potrebna za izgradnjo in učenje modela

Prikaz uporabe kode je na sliki 5.1, kjer na začetku ustvarimo nov model nevronske mreže, dodamo vhodni, polnopovezan sloj, kateremu določimo izhodne kot tudi vhodne dimenzije. Za tem slojem dodamo aktivacijski sloj z aktivacijo LeakyReLU. Z zanko dodamo n-skritih slojev, za vsakim določimo aktivacijo in delež izpadlih nevronov. Ustvarimo še izhodni sloj, ki ima enake izhodne dimenzije kot rezultati. Modelu na koncu definiramo funkcijo izgube in optimizacijski algoritem preden začnemo z učenjem.

5.3.1 Polnopovezan sloj (dense)

To je najbolj običajen sloj nevronske mreže. Ob klicu lahko dodamo več argumentov, med katerimi so najbolj pomembni units, activation in input_shape.

- input_shape - je pozitivno število, ki predstavlja dimenzijo vhodnega prostora. Običajno je ta argument naveden le v prvem polnopovezanem sloju. V naslednjih slojih ta argument ni potreben, saj mreža pričakuje, da bo dimenzija vhodnega prostora določenega sloja enaka izhodni dimenziji prejšnjega sloja.

- units - je pozitivno število, ki predstavlja dimenzijo izhodnega prostora.

- activation - je vrsta aktivacijske funkcije. Tu lahko navedemo zgolj osnovne vrste aktivacijskih funkcij (ReLU, tanh), ostale pa moramo dodati kot ločen sloj.

5.3.2 Izpadni sloj (dropout)

Uporaba plasti dropout poskrbi, da ni prekomernega prilagajanja (angl. overfitting) modela učnim podatkom. Dropout naključno izključi delež nevronov v sloju, tako da se njihove vrednosti ne prenesejo v naslednji sloj. Sloj dropout se lahko uporablja za vsakim polnopovezanim slojem. Ob klicu lahko dodamo argument rate, ki na intervalu med 0 in 1 pove, kolikšen delež nevronov bo izpuščen.

5.3.3 Aktivizacijski sloj (activation)

Če v dense sloju ni definirana vrsta aktivacijske funkcije, ali pa le-ta ni med enostavnimi (na primer LeakyReLU), se lahko aktivacijsko funkcijo definira v sloju Activation. Activation sprejme le en argument, in to je tip aktivacijske funkcije.

5.3.4 Sloj enodimenzionalne konvolucije (Conv1D)

Je glavni sloj enodimenzionalnih konvolucijskih nevronskih mrež. Ob klicu sprejme več argumentov, med pomembnejšimi pa sta filters in kernel_size. Filters je število izhodnih filtrov tega sloja, kernel_size pa določi dimenzije konvolucijskih filtrov.

5.3.5 Sloj enodimenzionalnega združevanja (MaxPooling1D)

Cilj sloja združevanja je, da podatke zgosti in s tem pospeši procesiranje podatkov. MaxPooling uporablja maksimalno združevanje podatkov, zahteva pa argument pool_size, ki definira velikost združevalnega okna. Večja ko je velikost združevalnega okna, manjše bodo izhodne dimenzije sloja.

5.3.6 Sloj sploščitve (flatten)

Ta sloj spremeni obliko podatkov, tako da jih lahko sprejme sloj dense.

5.4 Merjenje uspešnosti modelov

Uspešnosti delovanja dvorazrednih klasifikacijskih problemov se običajno prikazujejo z matriko napak (angl. confusion matrix). Ta prikazuje, koliko in kateri razredi so bili pravilno ali napačno napovedani glede na pravilen razred. Dvorazredna matrika napak vsebuje štiri polja. To so pravilno razvrščeni pozitivni primeri (angl. true positive) ali TP, napačno razvrščeni pozitivni primeri (angl. false negative) ali FN, napačno razvrščeni negativni primeri (angl. false positive) ali FP in pravilno razvrščeni negativni primeri (angl. true negative) ali TN. [13, 29]

Točnost (angl. accuracy) je mera, ki predstavlja razmerje med številom pravilno razvrščenih primerov in številom vseh primerov.

$$točnost = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

V primeru, ko opazujemo več kot dva razreda (v našem primeru 353), je smiselno opazovati tudi priklic in preciznost za vsak razred.

Preciznost (angl. precision) ali natančnost predstavlja razmerje med številom pravilno razvrščenih pozitivnih primerov in številom vseh pravilno razvrščenih primerov.

$$preciznost = \frac{TP}{TP + FP} \tag{5.2}$$

Priklic (angl. recall) predstavlja razmerje med številom pravilno razvrščenih pozitivnih primerov in številom vseh resnično pozitivnih primerov.

$$priklic = \frac{TP}{TP + FN} \tag{5.3}$$

Funkcija F, ki predstavlja povprečno mero preklica in natančnosti, se izračuna po formuli 5.4

$$F_1 = 2 * \frac{preciznost * priklic}{preciznost + priklic} \tag{5.4}$$

Ker je v naši podatkovni množici 353 razredov in bi zato konvergenčna matrika bila velikosti 353×353, bi jo bilo nesmiselno prikazati.

Za merjenje uspešnosti zgrajenega modela smo mu izmerili točnost, kar predstavlja utežen priklic, uteženo natančnost in funkcijo F_1. Utežena priklic in natančnost predstavljata povprečno vrednost priklica oziroma natančnosti vseh primerov testne množice.

5.5 Izbrane topologije in arhitekture nevronskih mrež

Zgradili smo dva tipa nevronskih mrež. Prva je bil globoka polnopovezana nevronska mreža, druga pa konvolucijska nevronska mreža. Ker število nevronov v skritih slojih ni nikjer točno določeno oziroma ni podane enačbe, ki bi veljala za vse podatke, smo se držali treh priporočil. [1]

- Število nevronov v skriti plasti naj bi bilo na intervalu med dolžinama vhodnih in izhodnih vektorjev.
- Število nevronov v skriti plasti naj bi bilo enako 2/3 velikosti seštevka velikosti vhodnih in izhodnih vektorjev.
- Število nevronov v skriti plasti naj bi bilo manj kot dvakrat večje od velikosti vhodnega vektorja.

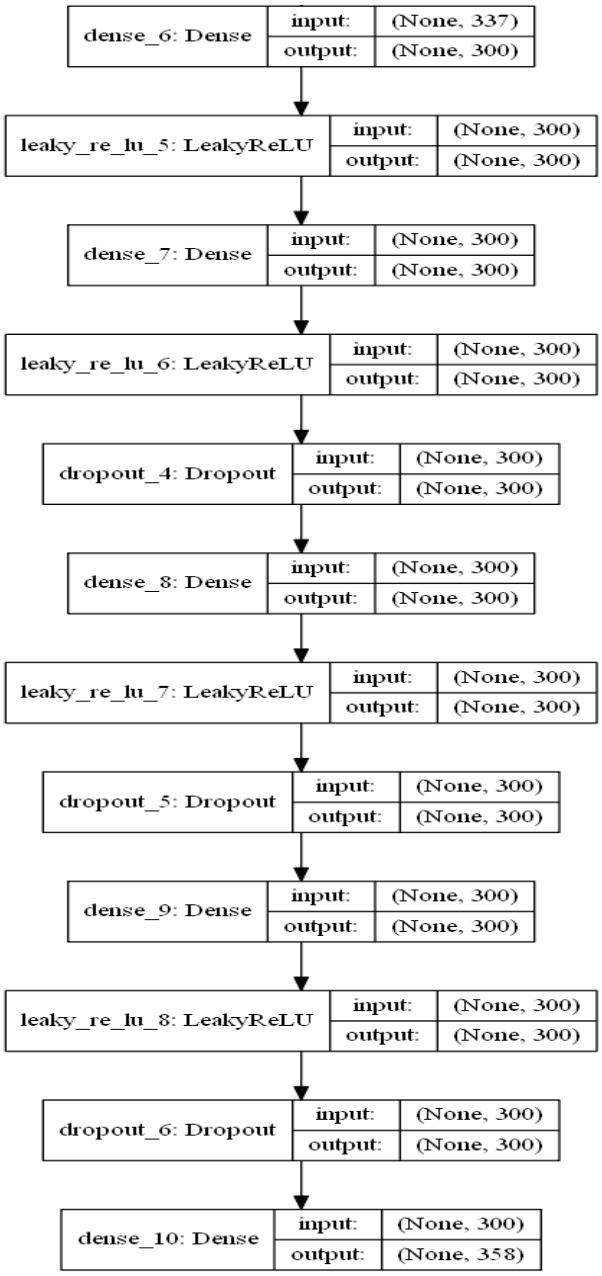
V polnopovezani nevronski mreži smo preverjali uspešnost mrež, ki imajo med 2 in 20 skritih slojev, v konvolucijski nevronski mreži pa med 2 in 15 konvolucijskih slojev ter 3 polnopovezane sloje. Ker konvolucijske nevronske mreže prepoznavajo vzorce med vhodnimi podatki, ki so si lokalno blizu, je bilo pomembno, da so atributi pred učenjem modela urejeni po podobnosti. V naši podatkovni množici preurejanje vrstnega reda atributov ni bilo potrebno, saj so atributi že urejeni po oddelkih. Vhodne vrstice so v mrežo prišle v obliki enodimenzionalnih vektorjev, nad katerimi sta se nato izvajala konvolucija in združevanje. Spodaj je podana slika modela globoke nevronske mreže 5.2.

5.5.1 Preprečevanje prekomernega prileganja

Do prekomernega prileganja (angl. overfitting) pride, ko se model preveč prilagodi podatkom iz učne množice. S tem doseže zelo visoko vrednost klasifikacijske točnosti na učni množici, a nizko točnost na testni množici.

Za preprečevanje prileganja smo uporabili sloje dropout in funkcijo keras.callbacks.EarlyStopping().

- Dropout: Sloji dropout izključijo naključne nevrone v posameznih slojih. S tem dosežejo, da se model težje prilega učnim podatkom. Običajno je delež izpadnih nevronov med 20% in 50%, mi smo izbrali delež 25%.
- EarlyStopping: Funkcija deluje tako, da ustavi učenje po tem, ko je rezultat opazovanega parametra k-krat zapored slabši od najboljšega. Če



Slika 5.2: Slika modela globoke nevronske mreže

je opazovani parameter funkcija izgube na testnih podatkih, parameter k pa je enak 5, se bo učenje modela ustavilo, ko bo rezultat funkcije izgube 5-krat zapored slabši od najboljšega.

5.5.2 Parametri

Skupni parametri, ki smo jih uporabili pri izgradnji modelov nevronskih mrež:

- delež dropout nevronov: 25%,
- aktivacijska funkcija skritih plasti: LeakyReLU,
- aktivacijska funkcija izhodne plasti: Sigmoid,
- število nevronov v skritih slojih: 300,
- optimizacijski algoritem: ADAM,
- funkcija izgube: kategorična križna entropija,
- število filtrov v konvolucijskem sloju: 128,

  število filtrov v konvolucijskih slojih ni točno določeno, pomembno pa je, da število ni preveliko ali premalo. Če je število filtrov preveliko, je učenje nevronske mreže časovno bolj potratno, če pa jih je premalo obstaja verjetnost, da v podatkih ne bodo razbrali pomembnih lastnosti.
- dimenzije filtrov: Ker so bili vhodni podatki enodimenzionalni, smo uporabljali tudi enodimenzionalne filtre. Dolžino filtrov smo nastavili na 3;
- velikost sloja enodimenzionalnega združevanja: 2,
- število iteracij učenja: se je spreminjalo, saj smo uporabili funkcijo keras.callbacks.Earlystopping(). Vrednost k smo nastavili na 5, opazovan parameter pa je bil rezultat funkcije izgube na testni množici. Število iteracij se je gibalo med 25 ter 60, njihovo največje število pa je bilo omejeno na 100.

5.6 Pristop k neuravnoteženim podatkom

Za delo z neuravnoteženimi podatki smo uporabili pythonovo knjižnico 'imblearn' [2], ki omogoča zmanjševanje oziroma povečanje števila podatkov in hibridne metode.

Namen uravnoteženja podatkov je bil naučiti model z večjim poudarkom na manj pogostih razredih. Pričakovali smo, da bo napovedna točnost nižja kot pri nespremenjenih podatkih, uspešnost napovedi manjšinskih razredov pa višja.

Poglavje 6

Rezultati

Rezultati so razdeljeni na pet delov. Prva tri analizirajo rezultate globokih in konvolucijskih nevronskih mrež, četrto je namenjeno upravljanju s neuravnoteženimi podatki, peto pa komentira učenje in uporabo nevronskih mrež na GPE ter CPE.

Vsi rezultati diplomskega dela so bili pridobljeni na računalniku s procesorjem AMD Ryzen 5 2600X hexa-core z 3.6GHz, grafično kartico GeForce GTX 1060 6GB in 8GB RAM. Operacijski sistem računalnika je bil Windows 10.

6.1 Globoka nevronska mreža

Za vsako od uporabljenih imputacij smo zgradili več modelov globokih nevronskih mrež. Zgrajene mreže so imele med 2 in 20 skritih slojev, med katerimi smo primerjali točnost, preciznost in funkcijo F_1 za posamezne imputacije (6.1a - 6.1e), nato pa točnosti vseh imputacijskih metod (6.1f).

Najvišjo točnost je dosegla imputacija k-najbližjih sosedov pri nevronski mreži s štirimi skritimi sloji. Točnost modela te mreže je znašala 45.7%, njegova povprečna natančnost 40% in funkcija F_1 33%. Prav vsi modeli, ki so imeli vhodne podatke imputirane z metodo k-najbližjih sosedov, so dosegli večjo točnost kot modeli drugih imputacij.

Glede na tip imputacije so si najboljše dosežene točnosti globokih nevronskih mrež sledile v naslednjem vrstnem redu: k-najbližjih sosedov, vstavljanje povprečnih vrednosti, vstavljanje konstantnih vrednosti, metoda MICE in kot najslabša vstavljanje vrednosti mediane. Posamezni rezultati za vsako od njih so prikazani v tabeli 6.1.

6.2 Konvolucijska nevronska mreža

Tako kot pri globoki nevronski mreži smo za vsako izmed uporabljenih imputacij zgradili model konvolucijske nevronske mreže. Vse konvolucijske nevronske mreže so imele 3 skrite sloje, razlikovale pa so se v številu konvolucijskih slojev. Števila le teh so bila med 1 in 14. Grafi točnosti, natančnosti in funkcije F_1 so prikazani na slikah 6.2a - 6.2e, kjer je tudi graf vseh točnosti (6.2f).

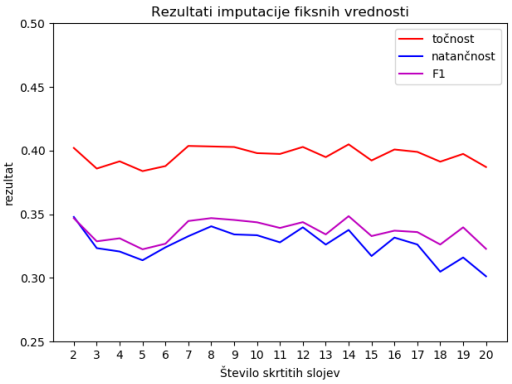
Najvišjo točnost je tudi tu dosegla mreža, katere podatki so bili imputirani z metodo k-najbližjih sosedov in je vključevala dva konvolucijska sloja. Napovedna točnost tega modela je bila 43%, njena natančnost in vrednost funkcije F_1 pa sta bili 37%.

Glede na najboljše točnosti posamezne imputacije si od najboljšega do najslabšega sledijo v naslednjem vrstnem redu: k-najbližjih sosedov, MICE, konstantne vrednosti, povprečne vrednosti in vrednosti mediane.

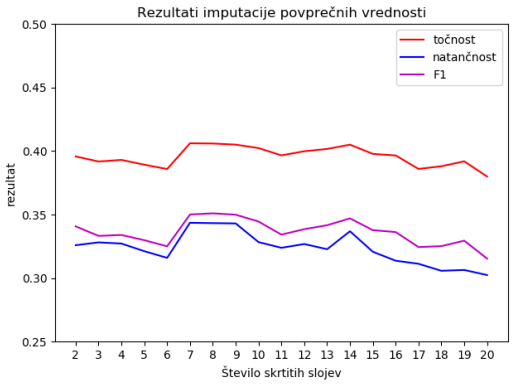
6.3 Primerjava topologij mrež

Spodaj je podana tabela najboljših rezultatov posameznih imputacij s podatki o vrsti modela in njegovo točnostjo. Pri obeh mrežah je najboljše rezultate proizvedla imputacija k-najbližjih sosedov in najslabše vstavljanje vrednosti mediane. Točnost najboljših mrež se je gibala med 37% in 45,7%. Iz tabele 6.1 je razvidno, da je globoka nevronska mreža dosegala boljše točnosti modelov v primerjavi z konvolucijsko nevronsko mrežo. Temu je tako, ker so naši podatki neslikovni, lokalna bližina podatkov pa pri njih ni tako po-

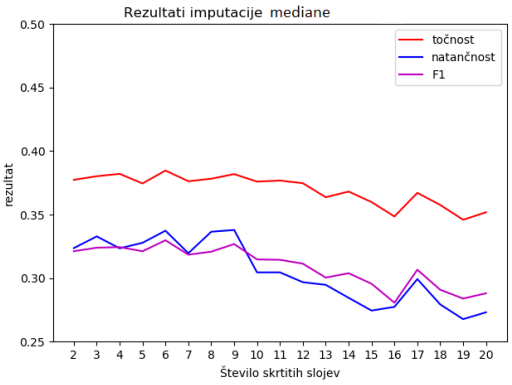
Slika 6.1: Grafi rezultatov globokih nevronskih mrež



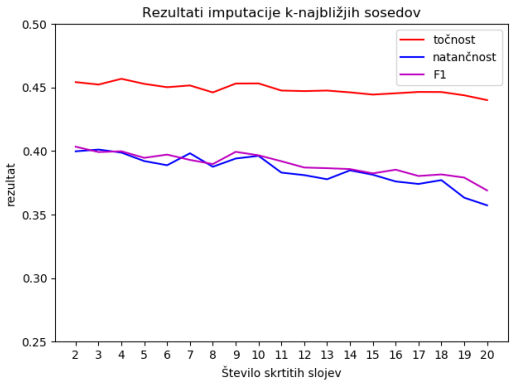
(a) Imputacija fiksnih vrednosti



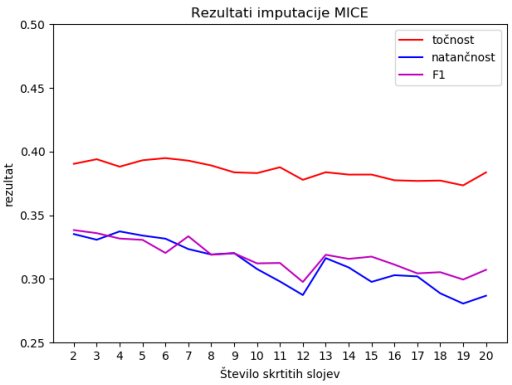
(b) Imputacija povprečnih vrednosti



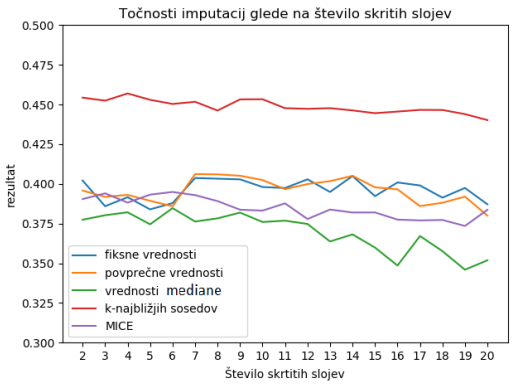
(c) Imputacija vrednosti mediane



(d) Imputacija k-najbližjih sosedov

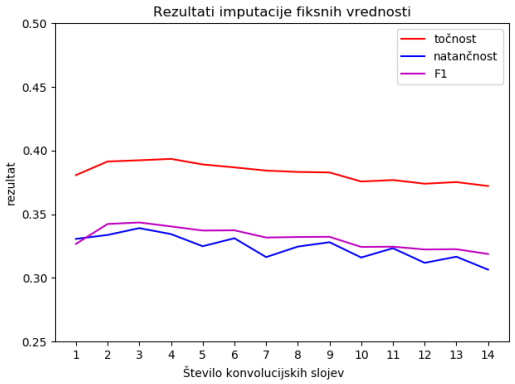


(e) Imputacija MICE

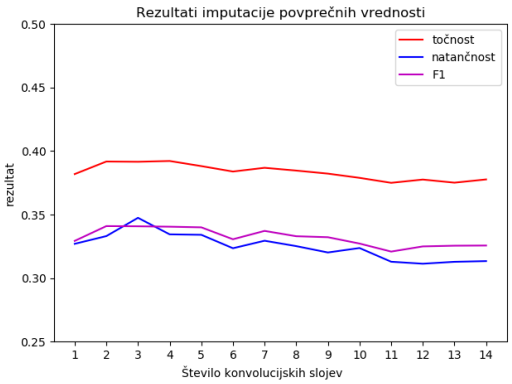


(f) Točnosti vseh imputacij

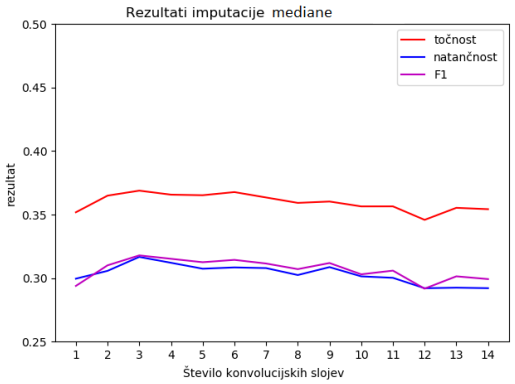
Slika 6.2: Grafi rezultatov konvolucijskih nevronskih mrež



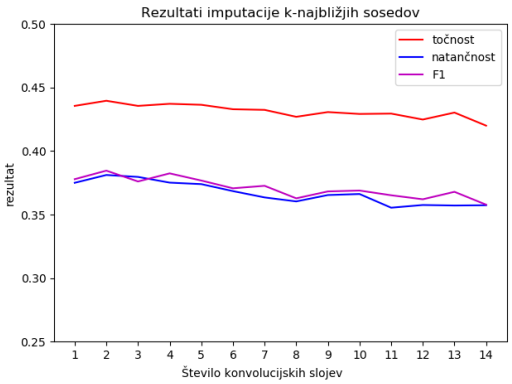
(a) Imputacija fiksnih vrednosti



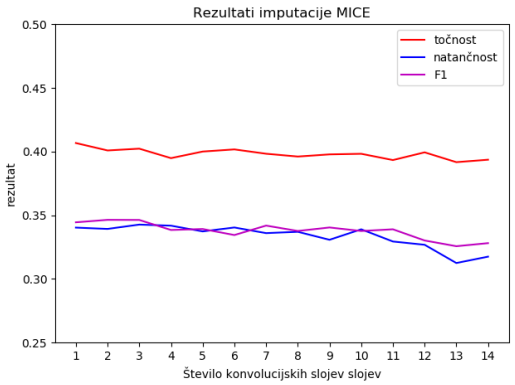
(b) Imputacija povprečnih vrednosti



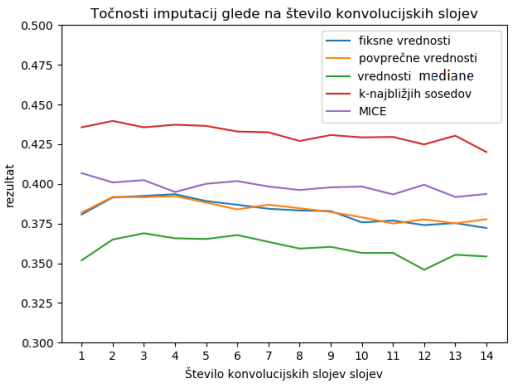
(c) Imputacija vrednosti mediane



(d) Imputacija k-najbližjih sosedov



(e) Imputacija MICE



(f) Točnosti posameznih imputacij

membna. Uporaba konvolucije in slojev združevanja zato ni učinkovita.

Vseeno sta se obe topologiji izkazali za približno enako dobri, njuna točnost je primerljiva s točnostjo, ki jo doseže algoritem ekstremnega gradientnega dviga (XGBoost). XGBoost doseže napovedno točnost 44%, deluje pa lahko brez imputacije, saj zna sam obravnavati manjkajoče podatke.

Tabela 6.1: Tabela najboljših rezultatov posameznih imputacij urejena od najbolj do najmanj točne

št.	topologija	imputacija	točnost (v odstotkih)
1.	globoka	k-najbližjih	45,7
2.	konvolucija	k-najbližjih	43,9
3.	konvolucija	MICE	40,6
4.	globoka	povprečne vrednosti	40,6
5.	globoka	fiksne vrednosti	40,5
6.	globoka	MICE	39,4
7.	konvolucija	fiksne vrednosti	39,3
8.	konvolucija	povprečne vrednosti	39,2
9.	globoka	mediana	38,5
10.	konvolucija	mediana	37,0

6.4 Rezultati upravljanja z neuravnoteženimi podatki

Za upravljanje z manjkajočimi podatki smo izbrali globoko nevronsko mrežo s štirimi skritimi sloji, njeni podatki pa so bili imputirani z metodo k-najbližjih sosedov. Odločili smo se, da zmanjšamo število podatkov, saj bi s povečanjem dobili preveliko množico. Najbolj pogost razred se namreč pojavi v 13214 vrsticah in če bi želeli pogostost vseh razredov povečati na enako število, bi število vrstic narastlo na več milijonov.

Število podatkov v učni množici smo zmanjšali do te mere, da so imeli najbolj pogosti razredi največ 1000 pojavitev v množici. Velikost učne množice se je s tem zmanjšala za polovico. Z zmanjšanjem ponovitev najpogostejših razredov smo dosegli, da je delež redkejših razredov v novi podatkovni množici relativno večji.

Zgrajen model je dosegel točnost 32%, njegova utežena preciznost je bila 28%, funkcija F_1 pa 26%. Čeprav so navedeni rezultati veliko slabši od tistih, ki so bili doseženi brez zmanjševanja števila podatkov, se je povečala občutljivost modela na tiste razrede, ki jih je v začetnem naboru manj.

Model brez upravljanja z neuravnoteženimi podatki je 100 najmanj zastopanih razredov napovedal v 3% primerov, model, ki pa je uporabil zmanjšano število večinskih razredov, je 100 najmanj zastopanih razredov napovedal v 12% primerov.

Metode za upravljanje z neuravnoteženimi podatki so dosegle slabše rezultate točnosti, preciznosti in funkcije F_1, a so bolje napovedovale manj zastopane razrede. Njihova uporaba bi bila smiselna, če bi bil naš cilj odkrivanje manj pogostih oz. redkih bolezni.

6.5 Analiza delovanja GPE in CPE

Učenje nevronskih mrež je potekalo na grafični kartici, saj je učenje na procesorju izrazito počasnejše. Učenje posameznega modela globoke nevronske mreže na procesorju traja vsaj 3-krat dlje od učenja na grafični kartici. Povprečen čas učenja modela globoke nevronske mreže je znašal 316 sekund (5 minut in 16 sekund), celoten čas izgradnje mrež za eno imputacijo pa je znašal več kot 8 ur. Razlog za to je, da smo uporabili 19 različnih globin in 5-kratno prečno preverjanje za vsako globino. Povprečen čas učenja modela konvolucijske nevronske mreže je znašal 11 minut in 33 sekund, celoten čas ene imputacije pa je znašal 13 ur in pol.

Za napovedovanje enega primera podatkov lahko uporabimo grafično kartico ali procesor. Pri modelu nevronske mreže s štirimi skritimi sloji je pro-

cesor za napovedovanje primera porabil 0,08 s, grafična kartica pa 0,34 s. Hitrost napovedi je pri obeh hitrejša od XGBoosta, ki za napoved potrebuje 0,5 s.

V primeru napovedovanja večjih skupin primerov pa se grafična kartica ponovno izkaže za učinkovitejšo. Pri napovedi 50000 primerov namreč procesor porabi 3,12 s, grafična kartica pa 1,48 s.

Čeprav se za napovedovanje večjih skupin primerov grafična kartica izkaže za učinkovitejšo, pa v praksi običajno potrebujemo napoved za posamezne primere, kjer je hitrejši procesor.

Poglavje 7

Sklepne ugotovitve

V diplomskem delu smo uporabljali globoke nevronske mreže za učenje iz podatkov medicinskih preiskav in napovedovanje diagnoz posameznih primerov. Raziskali in preverili smo točnost globokih nevronskih mrež na medicinskih podatkih, ki so imeli velik delež manjkajočih vrednosti ter so bili neuravnoteženi. Uporabili smo različne topologije in različne globine nevronskih mrež kot tudi mnoge načine imputacij. Pri topologiji, ki se je izkazala najbolje, smo preverili in komentirali še implementacijo metode za upravljanje z neuravnoteženimi podatki ter njene rezultate. Na koncu smo analizirali tudi učenje in uporabo nevronskih mrež na grafični kartici ter procesorju.

Med topologijami smo izbrali globoko polnopovezano usmerjeno nevronsko mrežo in konvolucijsko nevronsko mrežo. Za boljšo se je izkazala prva, razlog za to pa je dejstvo, da so podatki neslikovni in je uporaba konvolucije za njih manj smiselna. Opazili smo tudi, da napovedna točnost s povečevanjem skritih ali konvolucijskih slojev pada. Razlogov za to je lahko več, eden od njih je ta, da podatki niso zelo kompleksni in se lahko njihove medsebojne relacije identificira brez več dodatnih slojev.

V strojnem učenju je pomembno, da imamo dovolj velik in poln nabor podatkov. V primeru, da imajo podatki različne razpone vrednosti, jih je potrebno normalizirati na skupni interval. Če uporabljamo algoritem, ki pričakuje poln nabor podatkov, mi pa imamo v podatkovni množici manj-

kajoče podatke, je smiselno ugotoviti, kakšnega tipa so in jih temu ustrezno predprocesirati. Primere z manjkajočimi podatki lahko izbrišemo, vstavimo fiksne vrednosti ali pa izberemo način imputacije podatkov.

Če se posvetimo problemu neuravnoteženih podatkov, lahko pričakujemo, da bo točnost na testni množici močno padla. Kljub temu so metode uporabne v primeru, da je naš cilj povečati občutljivost modela na redkejše ali bolj pomembne razrede in s tem povečati priklic le-teh.

Za učenje nevronskih mrež je pomembno, da ga izvajamo na grafični kartici, ki je od procesorja veliko hitrejša pri učenju. Procesor je pri učenju slabši, a je hitrejši pri uporabi in napovedovanju posameznih razredov.

Pokazali smo, da se globoke nevronske mreže lahko primerjajo z drugimi algoritmi strojnega učenja. V primerjavi z XGBoostom so na medicinskih podatkih dosegle podobne točnosti, čas izgradnje modela je bil hitrejši, porabil pa je tudi manj pomnilnika.

Najpomembnejša slabost globokih nevronskih mrež je ta, da za delovanje potrebujejo polne podatke. Medicinske podatkovne množice običajno vsebujejo neznane vrednosti, saj vseh preiskav ne opravljajo pri vseh pacientih, zato je pri njih potrebno upravljanje z manjkajočimi podatki.

Literatura

[1] Heaton Research: the number of hidden layers. `https://www.heatonresearch.com/2017/06/01/hidden-layers.html`. Dostopano: 05.10.2018.

[2] Imbalanced learn. `https://pypi.org/project/imbalanced-learn/`. Dostopano: 16.10.2018.

[3] Slika konvolucijske nevronske mreže: mathworks. `https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html`. Dostopano: 16.10.2018.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association.

[5] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[6] Farid E. Ahmed. Artificial neural networks for diagnosis and survival prediction in colon cancer. *Molecular Cancer*, 4(1):29, Aug 2005.

[7] Paul D Allison. *Missing data*, volume 136. Sage publications, 2001.

[8] Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 20(1):40–49, 2011.

[9] Gustavo Batista and Maria Carolina Monard Monard. A study of k-nearest neighbour as an imputation method. *HIS*, 87(251-260):48, 2002.

[10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[11] François Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 7(8), 2015.

[12] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[13] Tom Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[14] Mingchen Gao, Ulas Bagci, Le Lu, Aaron Wu, Mario Buty, Hoo-Chang Shin, Holger Roth, Georgios Z Papadakis, Adrien Depeursinge, Ronald M Summers, et al. Holistic classification of ct attenuation patterns for interstitial lung diseases via deep convolutional neural networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(1):1–6, 2018.

[15] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19:263–282, 2009.

[16] John W. Graham. Missing data analysis: Making it work in the real world. *Annual Review of Psychology*, 60(1):549–576, 2009. PMID: 18652544.

[17] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

[18] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[19] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2(4):230–243, 2017.

[20] Esam Kamal Maried, Mansour Abdalla Eldali, Osama Omar Ziada, and Abdellatif Baba. A literature study of deep learning and its application in digital image processing. 06 2017.

[21] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[25] Geert Litjens, Clara I Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen-Van De Kaa, Peter Bult, Bram Van Ginneken, and Jeroen Van Der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6:26286, 2016.

[26] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, pages 1–11, 2017.

[27] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[28] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[29] David Martin Powers. Evaluation: from precision, recall and f-measure to ROC, informedness, markedness and correlation. 2011.

[30] Raul Rojas. *Neural Networks - A Systematic Introduction*. Raul Rojas, 1996.

[31] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.

[32] Rudy Setiono and Huan Liu. Neural-network feature selector. *IEEE transactions on neural networks*, 8(3):654–662, 1997.

[33] Devang Odedra Shankaracharya, Subir Samanta, and Ambarish S Vidyarthi. Computational intelligence in early diabetes diagnosis: a review. *The review of diabetic studies: RDS*, 7(4):252, 2010.

[34] Miss Mayuri S Shelke, Prashant R Deshmukh, and Vijaya K Shandilya. A review on imbalanced data handling using undersampling and oversampling technique. 3(4):444–449, 2017.

[35] Charalampos S Siristatidis, Charalampos Chrelias, Abraham Pouliakis, Evangelos Katsimanis, and Dimitrios Kassanos. Artificial neural networks in gynaecological diseases: Current and potential future applications. *Medical Science Monitor*, 16(10):RA231–RA236, 2010.

[36] Dayong Wang, Aditya Khosla, Rishab Gargeya, Humayun Irshad, and Andrew H Beck. Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*, 2016.