Strojno učenje u predviđanju sportskih rezultata

Borovec, Larisa

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Larisa Borovec

STROJNO UČENJE U PREDVIĐANJU SPORTSKIH REZULTATA

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Larisa Borovec

JMBAG: 1191234224

Studij: Organizacija poslovnih sustava

STROJNO UČENJE U PREDVIĐANJU SPORTSKIH REZULTATA

DIPLOMSKI RAD

Mentorica:

Doc. dr. sc. Oreški Dijana

Varaždin, rujan 2020.

Larisa Borovec

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je primjena odabranog algoritma strojnog učenja u analizi sportskih rezultata. U teorijskom dijelu rada opisuje se što je strojno učenje te kako i gdje se ono primjenjuje, opisuju se metode i algoritmi strojnog učenja uz naglasak na korištene algoritme. Opisuje se izabrani skup podataka i korišteni alati/tehnologije, a u praktičnom dijelu se isti primjenjuju nad odabranim skupom podataka. Skup podataka se odnosi na UFC borbe od 1993. do 2019. godine. Tehnologije koje se korise za izradu, treniranje, validaciju i korištenje prediktivnog modela su Python programski jezik, Tensorflow i Keras te online platforma za strojno učenje BigML. Cilj rada je pokazati na koji način se model strojnog učenja implementira te kakve rezultate daje u domeni sportskih rezultata.

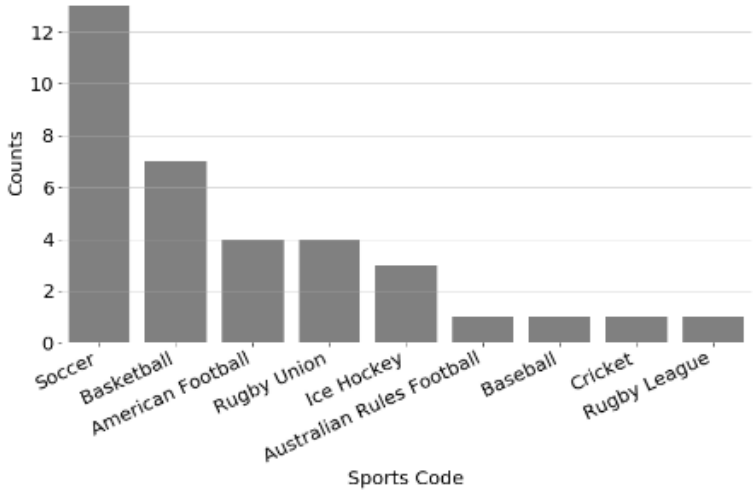
Ključne riječi: strojno učenje; nadzirano učenje; neuronska mreža; Python; Keras; sport

Sadržaj

1. Uvod

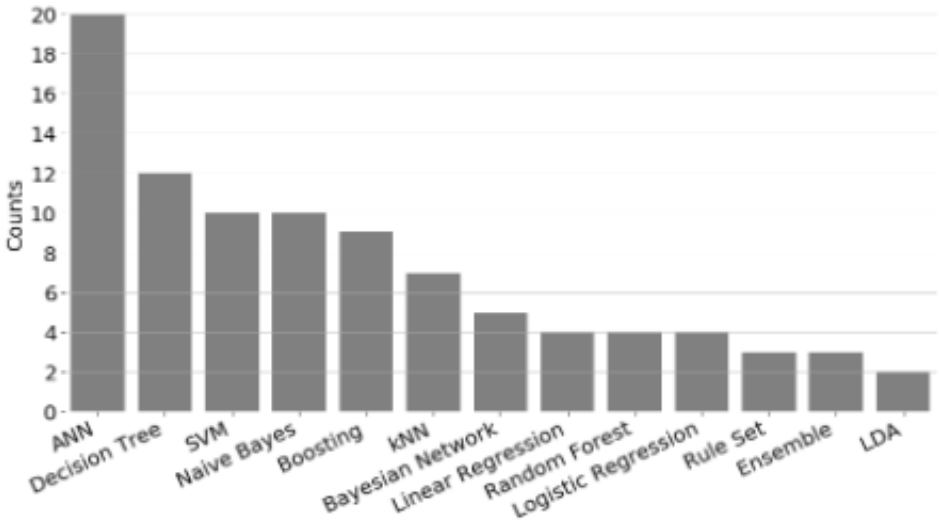
Sport je oduvijek pratio svakodnevni život ljudi – od svojevrsnog duhovnog rituala preko vojne obuke do zabave. Uz sam sport, uvijek je ruku pod ruku išlo i sportsko klađenje, još od vremena antičkih Grka koji su predstavili Olimpijske igre i najranije zapise o klađenju na atletskim natjecanjima. Od Grka se sportsko klađenje proširilo na stari Rim, gdje je ono čak i legalizirano. Rimljani su se najviše kladili na igre gladijatora. Iako su igre gladijatora zaustavljene, kockanje je preživjelo i širilo se na druga kraljevstva i sportove. Tijekom srednjovjekovnih vremena neki su vjerski vođe pokušavali donijeti zakone koji to zabranjuju. To je potaknulo sportsko klađenje u podzemlju, gdje je nastavilo postojati, pa čak i rasti kad su novi sportski događaji predstavljeni svijetu. Kasnije u povijesti kockanje je postalo vrlo popularno u Engleskoj u obliku klađenja na konjske utrke. Englezi su tu praksu proširili na ostatak svijeta, posebno u SAD, gdje je to brzo postalo omiljena zabava mnogih ljudi. Sportsko klađenje veoma je popularno i danas, posebice u cijeloj Europi koja je postala najveće tržište sportskih kladionica, čime sportsko klađenje nije više samo zabavna aktivnost, već i posao vrijedan više milijardi dolara. Razvojem informacijskih tehnologija i pojavom interneta sportsko klađenje mahom se raširilo i dalje raste iz godine u godinu. Internet je omogućio bolju dostupnost informacija, klađenje iz ugodnosti vlastitog doma, klađenje u živo i slično. [1]

Predviđanje sportskih rezultata zanimljiv je i izazovan problem zbog nepredvidive prirode sporta i naizgled beskonačnog broja potencijalnih čimbenika koji mogu utjecati na rezultate, a za ovakav tip problema najviše su zainteresirani sudionici sportskog klađenja. Prema Bunker, Susnjak [2] tema predviđanja sportskih rezultata korištenjem strojnog učenja sve je popularnija, najviše se analiziraju timski sportovi (slika 1), a najzastupljeniji algoritmi za predviđanje sportskih rezultata su neuronske mreže i stabla odlučivanja (slika 2).



Slika 1: Broj istraživanja prema sportu [2]

Točnost predviđanja samih modela za timske sportove iznosi oko 70%, pri čemu su neki modeli malo ispod 60%, a neki malo iznad 80%, ovisno o sportu. Postoji nekoliko istraživanja vezanih uz individualne sportove poput konjskih utrka, plivanja, golfa i tenisa, pri kojima točnost predviđanja modela iznosi oko 65%. [2]



Slika 2: Broj istraživanja prema algoritmu strojnog učenja [2]

Popularnost sporta i sportskog klađenja te razvoj informacijskih tehnologija, posebice popularnost strojnog učenja, motivacija su za izradu ovog rada, ne toliko zbog samog sporta i klađenja, već znatiželje može li se i u kojoj mjeri uporabom strojnog učenja predvidjeti rezultat neke utakmice ili meča. Kako su individualni sportovi manje popularni za istraživanje nego timski sportovi, cilj ovog rada je pronaći skup podataka za individualni sport i kreirati model strojnog učenja te ispitati u kojoj mjeri je moguće predvidjeti rezultate. Za samo kreiranje čim točnijeg modela potrebno dobro poznavati domenu, odnosno sam sport kako bi se znalo što bolje odrediti koje su to važne značajke koje doprinose točnijem predviđanju. Pretpostavlja se da sam model može postići točnost predviđanja oko 60% zbog manjka poznavanja domene. Za algoritam strojnog učenja odabrana je neuronska mreža jer je upravo ona najpopularniji algoritam za predviđanje sportskih rezultata. Rad je podijeljen u dva dijela, u prvom dijelu dan je pregled teorijskih koncepata vezanih uz strojno učenje i neuronske mreže, a nakon toga prikazana je sama izvedba praktičnog dijela, odnosno kreiranja neuronske mreže uz objašnjene korake i rezultate te je na kraju dan zaključak.

2. Metode i tehnike rada

Rad se odrađuje kroz dva dijela, teorijski i praktični dio. Cilj teorijskog dijela je pobliže opisati što je strojno učenje te kako i gdje se ono može primijeniti, odnosno dati uvod za bolje razumijevanje praktičnog dijela. Literatura koja se koristi rezultat je pretraživanja internetskih stranica i repozitorija prema ključnim riječima - umjetna inteligencija (eng. *artificial intelligence*) i strojno učenje (eng. *machine learning*) te pojmovi vezani uz njih. Za praktični dio, odnosno prikaz izrade modela i primjene strojnog učenja, pretraživanjem interneta pronađen je i odabran skup podataka vezan uz UFC borbe od 1993. do 2019. godine [3]. Zbog rastućeg trenda uporabe umjetne inteligencije posljednjih godina i mogućnosti za zaposlenje te prilike da se usvoje nova znanja i vještine, za izvedbu praktičnog dijela odabran je pristup izgradnje od nule umjesto korištenja dostupnih online alata za strojno učenje. Model kreiran u online alatu korišten je za usporedbu, odnosno određivanje u kojoj mjeri je praktični rad autora uspješan u odnosu na automatizacijom kreirani model. U nastavku je dan pregled svih korištenih tehnologija i alata.

Programski jezik koji je odabran za izradu je Python 3.8.3 zbog velikog broja biblioteka koje su specijalizirane za strojno učenje, npr. TensorFlow, Teano, Keras, PyTorch, Scikit-learn. Prednost ovog programskog jezika je jednostavnost, pristup bibliotekama, fleksibilnost, neovisnost o platformi te široka zajednica.

Anaconda Navigator je grafičko sučelje (eng. *graphical user interface*, GUI) koje pruža potporu Conda sustavu. Conda je sustav upravljanja paketima otvorenog koda i sustav upravljanja okruženjem. Brzo instalira, pokreće i ažurira pakete i njihove ovisnosti. [4] Ovaj alat korišten je za instalaciju svih potrebnih biblioteka i pokretanje uređivača VS Code.

VS Code, odnosno Visual Studio Code je besplatni uređivač (*editor*) izvornog koda koji je izradio Microsoft. VS Code odabran je zbog jednostavnosti korištenja.

TensorFlow je cjelovita platforma otvorenog koda za strojno učenje, a koristi se za izradu aplikacija strojnog učenja. Omogućava jednostavno kreiranje i treniranje modela strojnog učenja korištenjem intuitivnih sučelja (pr. Keras) s brzim izvršavanjem iteracija modela što omogućava jednostavno uklanjanje grešaka. [5]

Keras je API (eng. *application programming interface*), odnosno sučelje koje se koristi za izradu neuronskih mreža. Dizajniran je kako bi omogućio brzo eksperimentiranje s (dubokim) neuronskim mrežama, fokusira se na to da bude lak za uporabu (eng. *user-friendly*), modularan i proširiv. Slijedi najbolje prakse kako bi korisnici sa čim manje truda kreirali modele te pruža jasne poruke o pogreškama te daje ideje kako ih promijeniti. Ima opsežnu dokumentaciju i vodiče za programere što ga čini primamljivim za kreiranje rješenja. [6]

Pandas je Python biblioteka koja pruža brzu, fleksibilnu i jednostavnu analizu i manipulaciju podacima. Izgrađena je na paketu Numpy, a njezina se ključna struktura podataka naziva DataFrame. DataFrame-ovi omogućuju pohranjivanje i manipulaciju tabličnim podacima. [7] Korištena je za pripremu skupa podataka nad kojim se kasnije model trenirao, validirao i testirao, a sama upotreba biblioteke vrlo je jednostavna i

U samom rješenju još je korištena biblioteka NumPy za upravljanje poljima podataka te Matplotlib [8] za kreiranje grafičkih prikaza vezanih uz treniranje modela.

Grafički prikazi neuronskih mreža kreirani su u NN SVG alatu [9].

Za automatiziranu izradu modela neuronske mreže korištena je sveobuhvatna online platforma za strojno učenje BigML. BigML nudi izbor robusnih inženjerskih algoritama za strojno učenje koji dokazano rješavaju probleme iz stvarnog svijeta primjenom jedinstvenog, standardiziranog okvira. Pruža trenutačno strojno učenje nadohvat ruke. Web sučelje vrlo je intuitivno i jednostavno za korištenje, kreiranje modela je brzo i efikasno, a za uporabu platforme potrebno je samo prijaviti se putem e-pošte. [10].

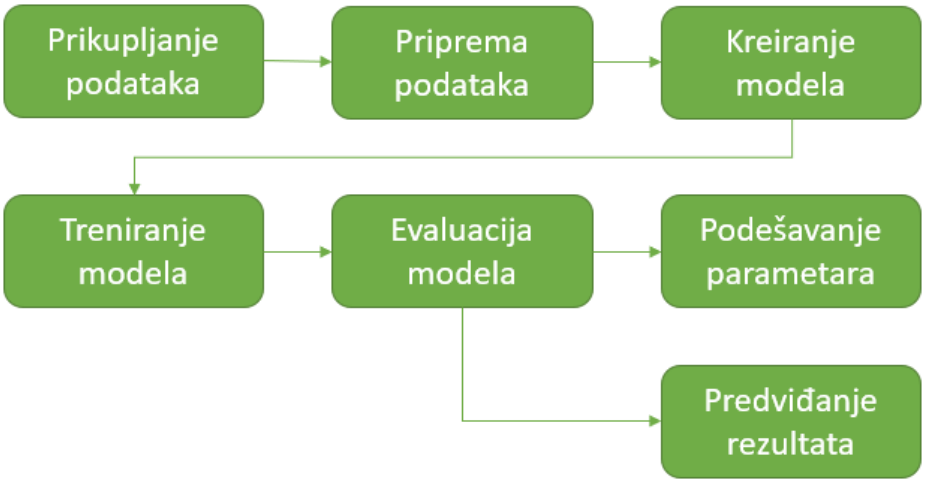
3. Strojno učenje

Strojno učenje oblik je umjetne inteligencije. Umjetna inteligencija dio je računalne znanosti koja se bavi simulacijom inteligentnog ponašanja kod računala, odnosno mogućnošću računala da imitira ljudsko ponašanje. Jedan od oblika umjetne inteligencije je skup ako-onda (eng. *if-then*) izjava koje su eksplicitno programirane od strane čovjeka. Takav oblik umjetne inteligencije naziva se ekspertni sustav. Ono što razlikuje strojno učenje od ekspertnih sustava je mogućnost modela da sam sebe dinamički modificira bez ljudske intervencije. [11]

Strojno učenje može se u širem smislu definirati kao skup metoda koje temeljem iskustva unaprjeđuju izvođenje ili donose točna predviđanja. Iskustvo se u ovom slučaju odnosi na prošle informacije dostupne učeniku (stroju) pri čemu su te informacije elektronički skup podataka koji je prikupljen i dostupan za analizu. Kvaliteta i količina danih podataka direktno utječu na rezultat, npr. dan je konačan skup dokumenata koji imaju označen tip poput narudžbenice, ulazne fakture, izlazne fakture i slično. Ukoliko je skup podataka koji je dostupan za analizu velik, stroju će biti omogućeno da bolje nauči razlikovati dokumente. Istovremeno, ukoliko se desi da dokumenti nisu ispravno označeni (pr. neke narudžbenice su označene kao dostavnice) ili pak postoji tip dokumenta koji nije obuhvaćen setom podataka za analizu (pr. otpremnica), stroj neće znati ispravno zaključiti o kojem se tipu radi prilikom predviđanja. Glavni izazov strojnog učenja jest dizajniranje učinkovitih i preciznih algoritama predviđanja. Važne mjere kvalitete tih algoritama su vremenska i prostorna složenost, ali i složenost uzorka, odnosno broj razmatranih konceptualnih klasa te veličina uzorka za treniranje algoritma. Sam cilj strojnog učenja je od skupa prikupljenih podataka doći do zaključka, shvaćanja vezanog uz dane podatke.

Problemi koje strojno učenje može rješavati su brojni – klasifikacija dokumenata, obrada prirodnog jezika (interakcija između računala i ljudi korištenjem jezika ljudi), obrada govora, prepoznavanje i sinteza govora, identifikacija govornika, optičko prepoznavanje znakova, prepoznavanje i identifikacija objekta, otkrivanje lica, predviđanje funkcije proteina, analiza mreža gena i proteina, otkrivanje kartičnih prijevara, upadi u mrežu, medicinska dijagnostika, sustav preporuka i brojni drugi. Zbog široke lepeze problema koje strojno učenje može riješiti, moguće ga je primijeniti gotovo u svim područjima, a najviše se primjenjuje u područjima medicine, osiguranja, farmacije, prodaje, bankarstva i financija, naftnih kompanija te transporta. [12]

Proces kreiranja rješenja, odnosno primjena algoritma strojnog učenja za odabrani problem sastoji se od nekoliko koraka prikazanih na slici 3.



Slika 3: Proces kreiranja modela strojnog učenja [rad autora]

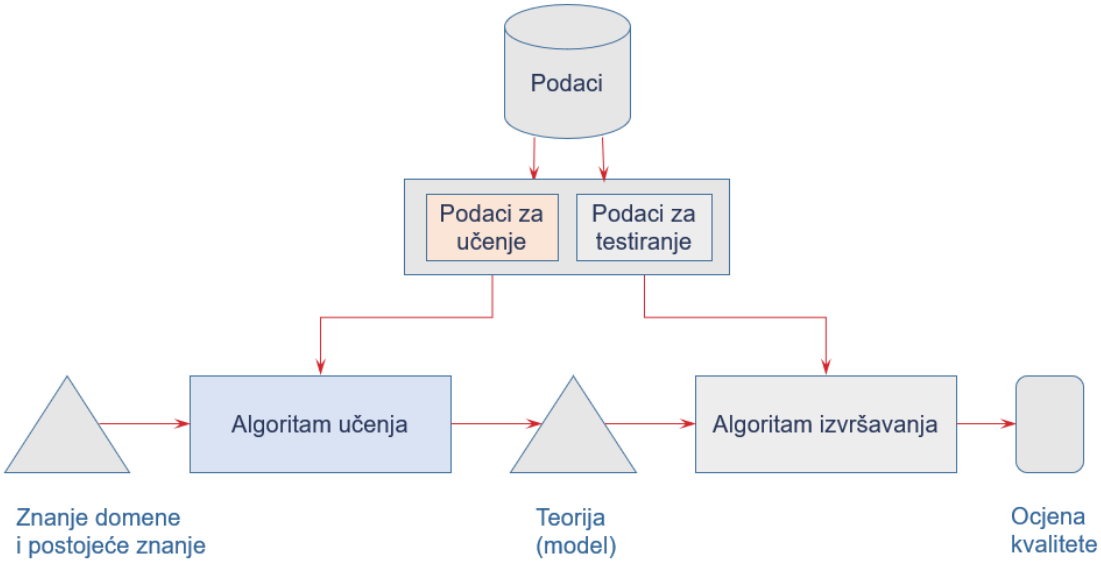
Nakon što se definira problem, potrebno je prikupiti podatke za analizu. Prije samog prikupljanja poželjno je odrediti kakvi podaci moraju biti kako bi predstavljali reprezentativan skup koji obuhvaća sve moguće klase rješenja. Važno je odrediti koje značajke doprinose rješenju kako samo prikupljanje podataka ne bi bilo preskupo i preopširno. Ukoliko već imamo dostupne podatke, potrebno je izdvojiti značajke koje doprinose kako se model ne bi nepotrebno opterećivao. Poželjno je da skup ima što više podataka, no to često nije moguće (podatke je potrebno prikupljati u više iteracija, nedostupnost podataka, podaci na različitim mjestima). Ovisno o odabranom algoritmu, potrebno je podatke dodatno pripremiti (npr. kodiranje kategorijskih varijabli, normalizacija numeričkih varijabli). Podatke je zatim potrebno podijeliti na skupove za treniranje i testiranje. Prilikom odabira algoritma strojnog učenja potrebno je obratiti pažnju na [13]:

- veličinu skupa podataka za treniranje modela – ukoliko skup podataka ima mali broj opažanja i velik broj značajki, preporuča se odabrati linearnu regresiju ili Bayesovu mrežu
- točnost i mogućnost interpretacije rezultata – ukoliko je poželjna mogućnost jednostavne interpretacije rezultata, preporuča se linearna regresija, dok veća točnost rezultata za sobom povlači fleksibilnost modela, što znači da će rezultate biti teže interpretirati (pr. putem dijagrama)
- brzinu ili vrijeme treniranja – veća točnost modela za sobom povlači dulje vrijeme treniranja, kao i veća količina podataka. Linearna regresija i Bayesove mreže jednostavni su za implementaciju i brzi, dok neuronske mreže i nasumične šume zahtijevaju puno vremena za treniranje
- linearnost – neki od algoritama rade temeljem pretpostavke da se klase podataka mogu odvojiti ravnom linijom – ukoliko su podaci linearni, takvi

algoritmi rade prilično dobro, dok je u suprotnom potrebno odabrati nasumičnu šumu ili neuronsku mrežu

- klasifikacija podataka – ukoliko podaci nisu klasificirani, odabire se algoritam nenadziranog učenja, dok se u suprotnom odabire algoritam nadziranog učenja

Nakon odabira algoritma, ovisno o samom algoritmu definiraju se potrebni parametri te se izrađuje model. Model se zatim trenira skupom podataka za treniranje čime model zapravo uči kako bi mogao prilikom testiranja dati željene izlazne vrijednosti. Nakon treniranja model se testira te se procjenjuje točnost predviđanja na testnim podacima. Temeljem evaluacije provode se promjene nad modelom ukoliko je potrebno te se model ponovo trenira. Na kraju se model koristi za predikciju. U nastavku je na slici 4 prikazan proces učenja samog modela: prikupljeni podaci podijele se u dva skupa, jedan za treniranje, odnosno učenje i drugi za testiranje. Uporabom znanja o domeni i postojećih znanja kreira se algoritam učenja koji korištenjem skupa podataka za učenje omogućava modelu da uči i kreira teoriju o podacima. Nakon toga se korištenjem algoritma izvršavanja vrši testiranje naučene teorije modela te se ocjenjuje kvaliteta naučenog.



Slika 4: Proces učenja [14]

Kako strojno učenje ima široko polje primjene, postoji nekoliko metoda odnosno scenarija učenja čiji pregled je dan u nastavku. Nakon toga dan je pregled najpoznatijih algoritama strojnog učenja.

3.1. Metode strojnog učenja

Zbog svoje popularnosti i širokog područja primjene, konstantno se razvijaju nove metode i algoritmi strojnog učenja kako bi se problemi rješavali brže i efikasnije te kako bi se

doskočilo raznim problemima vezanim uz podatke nad kojima algoritmi uče (manji skup podataka, nepotpuni skupovi, veliki skupovi). Četiri najčešće prihvaćene metode strojnog učenja su nadzirano učenje (eng. *supervised learning*), nenadzirano učenje (eng. *unsupervised learning*), polunadzirano učenje (eng. *semisupervised learning*) te učenje nagrađivanjem (eng. *reinforcement learning*), koje su detaljnije objašnjene u nastavku prema Li [15].

3.1.1. Nadzirano učenje

Algoritmi nadziranog učenja uče temeljem skupa podataka pri kojima je željeni rezultat za određene ulazne vrijednosti poznat, npr. algoritam određuje je li na slici mačka ili pas te se ulazni podaci sastoje od parova koji sadrže sliku i oznaku (eng. *label*) je li na slici pas ili mačka. Algoritam uči na način da sam pretpostavi što je na slici i zatim svoj rezultat uspoređuje s danim oznakama te u skladu s rezultatima usporedbe modificira model. Na taj način kreira funkciju koja novim, nepoznatim primjerima daje oznaku temeljem generaliziranja provedenog nad podacima za učenje. Nadzirano učenje koristi se kada povijesni podaci predviđaju vjerojatne buduće događaje, npr. koji će klijent osiguranja vjerojatno podnijeti odštetni zahtjev. Problemi koje rješava nadzirano učenje mogu se podijeliti u sljedeće kategorije:

- Klasifikacija – koristi se kada je značajka koja se predviđa kategorijska varijabla poput ranije spomenutog primjera sa slikama i psom i mačkom. Ukoliko postoje samo dvije ciljne oznake, tada se radi o binarnoj klasifikaciji, a kada ih je više, radi se o višerazrednoj klasifikaciji.
- Regresija – koristi se prilikom predviđanja kontinuiranih vrijednosti, primjerice kada se predviđaju cijene stanova ili vrijeme potrebno da se izvrši neki proces u proizvodnji

Linearna regresija i nasumična šuma primjer su algoritama nadziranog učenja.

3.1.2. Nenadzirano učenje

Nenadzirano učenje koristi se kada skup ulaznih podataka nije označen, odnosno modelu se ne može dati do znanja koji je „točan odgovor“ već on mora sam shvatiti što se prikazuje. Cilj ovog načina učenja je otkriti strukturu u podacima, a dobro funkcionira na transakcijskim podacima, npr. otkrivanje segmenata kupaca. Ovaj tip učenja može se nadalje podijeliti u dvije skupine:

- Klasteriranje – grupiranje podataka tako da podaci u jednoj skupini, odnosno klasteru, budu prema nekim kriterijima sličniji od onih u drugim skupinama, a koristi se kako bi se skup podataka podijelio na nekoliko manjih grupa

- Smanjenje dimenzija – koristi se za smanjenje broja varijabli (atributa) koji se razmatraju prilikom modeliranja. U mnogim slučajevima sirovi podaci imaju mnoštvo atributa koji su često suvišni ili redundantni i ne pridonose rješavanju problema. Smanjivanje dimenzionalnosti pomaže u pronalaženju pravog odnosa među podacima.

3.1.3. Polunadzirano učenje

Polunadzirano učenje koristi se za rješavanje istih problema kao i nadzirano učenje, a primjenjuje se kada je skup dostupnih podataka neoznačen ili djelomično označen – neobilježeni podaci često su jeftiniji za prikupljanje te je potrebno manje truda za njihovo prikupljanje. Ukoliko je skup neoznačen, potrebno je označiti klase koje rješenje može poprimiti. Algoritam zatim koristi taj manji skup označenih podataka, uči nad njima te naučeno koristi za označavanje neoznačenog dijela skupa podataka. Ovaj tip najčešće se koristi kada su troškovi povezani s označavanjem ulaznog skupa podataka previsoki da bi omogućili potpuno obilježen proces učenja, a primjer toga je prepoznavanje lica na web-kameri – na manjem skupu slika označe se lica kako bi algoritam taj skup koristio za daljnje učenje i prepoznavanje na ostatku slika iz prikupljenog skupa.

3.1.4. Učenje nagrađivanjem

Učenje nagrađivanjem analizira i optimizira ponašanje agenata na temelju povratnih informacija iz okoline. Algoritmi metodom pokušaja i pogreške iskušavaju različite scenarije kako bi otkrili koje radnje donose najveću nagradu, umjesto da im se eksplicitno kaže koje radnje treba poduzeti. Ima tri komponente: agent (učenik ili donositelj odluke), okruženje (sve s čim agent komunicira) i radnje (što sve agent može učiniti). Često se koristi za robotiku, igre i navigaciju. Cilj učenja je da agent prepozna i odabere radnje koje maksimiziraju očekivanu nagradu tokom određenog vremena te da na taj način nauči najbolju politiku kako bi čim brže stigao do određenog cilja.

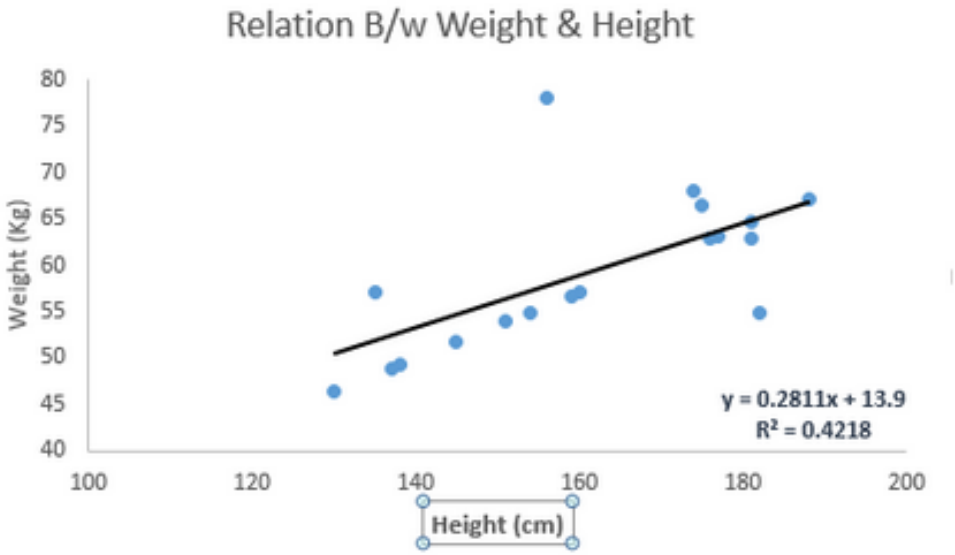
3.2. Algoritmi strojnog učenja

Srž strojnog učenja su algoritmi strojnog učenja. Osim što se razlikuju po načinu na koji uče, razlikuju se i po tome što se koriste za rješavanje različitih problema. Prema Ray [16] najčešće korišteni algoritmi su linearna regresija (eng. *Linear Regression*), logistička regresija (eng. *Logistic Regression*), stablo odlučivanja (eng. *Decision Tree*), neuronske mreže (eng. *Neural Network,* NN), potporni vektori (eng. *Support Vector Machine*, SVM), Bayesova mreža (*Naive Bayes*), k-najbliži susjedi (eng. *k-Nearest Neighbors*, kNN), K-sredine (eng. *K-Means*),

nasumična šuma (eng. *Random Forrest*), algoritam smanjenja dimenzionalnosti (eng. *Dimensionality Reduction Algorithms*) i algoritmi pojačavanja gradijenta (eng. *Gradient Boosting algorithms*). U nastavku je dan pregled odabranih algoritama.

3.2.1. Linearna regresija

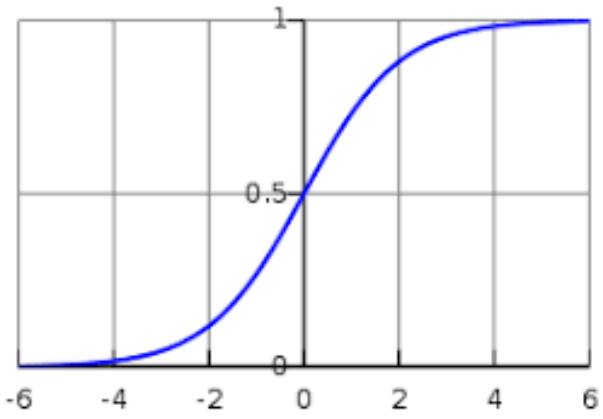
Linearna regresija izražava odnos između ulaza x i izlaza y – za svaku promjenu vrijednosti x, y će se proporcionalno promijeniti. Ukoliko se radi sa samo jednom ulaznom varijablom tada je to jednostavna linearna regresija, a u slučajima s više ulaznih varijabli radi se o višestrukoj linearnoj regresiji. Koristi se za procjenu stvarnih vrijednosti (cijena nekretnina, broj poziva, ukupna prodaja) temeljem kontinuiranih varijabli. Cilj linearne regresije je kreirati liniju koja najbolje opisuje odnos nezavisnih i zavisne varijable. Predstavljena je linearnom jednadžbom: $\hat{y} = a \cdot x + b$, a za bolje razumijevanje prikazan je jedan graf linearne regresije na slici 5 u nastavku. [16]



Slika 5: Graf linearne regresije [16]

3.2.2. Logistička regresija

Logistička regresija zapravo nije regresija već algoritam za klasifikaciju. Koristi se za procjenu diskretnih vrijednosti, poput binarnih vrijednosti (0 ili 1), da ili ne, točno ili netočno, istina ili neistina. Radi na način da predviđa vjerojatnost pojave događaja temeljem logističke funkcije, što znači da kao rezultat daje vrijednosti između 0 i 1, a kao ulaz može primati jedan ili više parametara. Grafički prikaz dan je na slici 6.

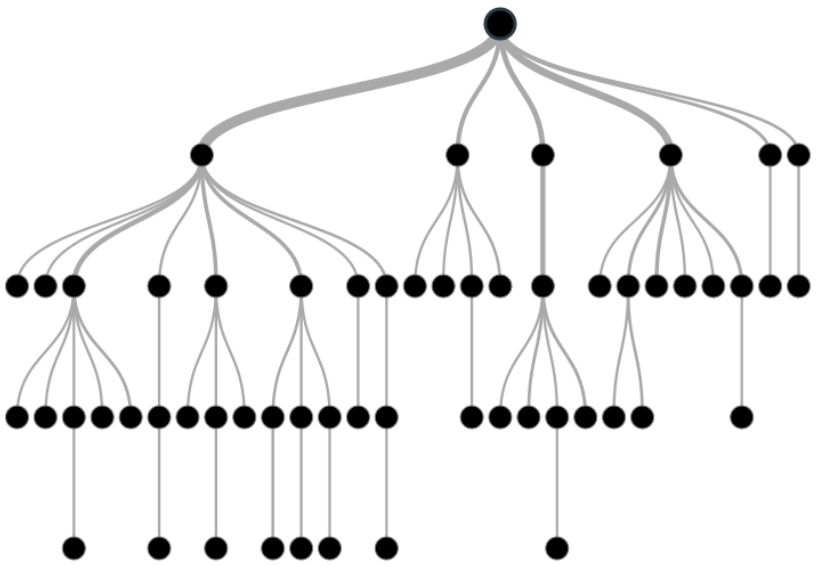


Slika 6: Graf logističke funkcije [17]

Ovaj algoritam često se koristi za klasteriranje pri nenadziranom učenju, a često se koristi i kao funkcija aktivacije u posljednjem koraku algoritma. [17]

3.2.3. Stablo odlučivanja

Stablo odlučivanja niz je čvorova, odnosno usmjereni graf koji započinje jednim čvorom kojeg nazivamo korijenom stabla i proteže se na brojne čvorove – listove koji predstavljaju kategorije koje stablo može klasificirati. Odluku donose ispitivanjem značajki i odbacivanjem mogućnosti koje ne vrijede za neku određenu situaciju, a sam proces može se predstaviti postavljanjem pitanja koja algoritam pita za određenu instancu koja se promatra, a skup podataka daje odgovore na pitanja te se na taj način kreće čvorovima, odnosno granama u stablu. Prikaz stabla odlučivanja dan je slikom 7.

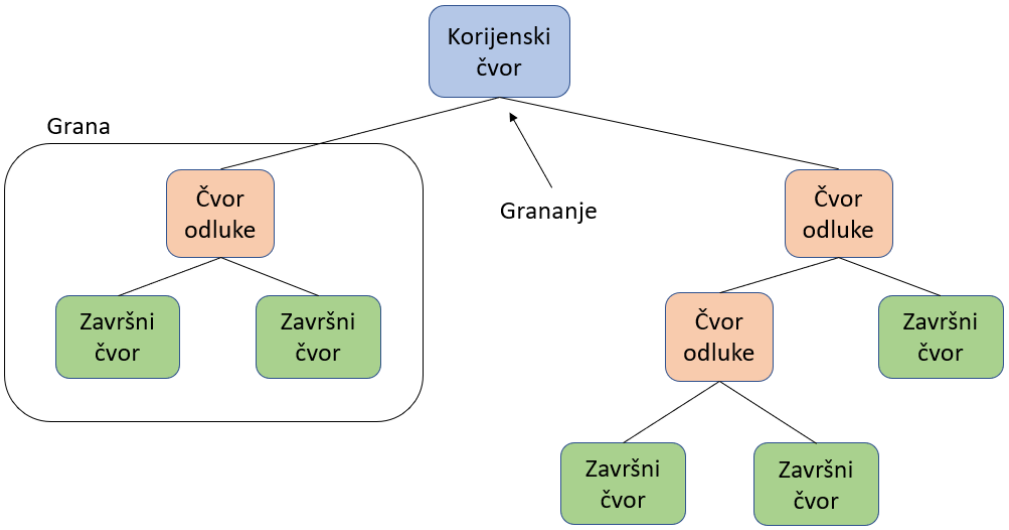


Slika 7: Stablo odlučivanja [18]

Stablo odlučivanja sastoji se od sljedećih elemenata:

- Korijenski čvor – predstavlja cjelokupnu populaciju koja se analizira, a od njega se populacija dijeli prema različitim značajkama te se svaka sljedeća podskupina dijeli na narednom čvoru odluke sve do listova
- Grananje – postupak podjele čvora na dva ili više pod-čvora
- Čvor odluke – čvor koji se grana
- Završni (terminalni) čvor ili list – čvor koji se ne grana, završni čvor
- Rezidba – uklanjanje odnosno rezanje pod-čvorova nadređenog čvora – stablo se širi grananjem i smanjuje rezidbom
- Grana ili podstablo – dio stabla koji se može gledati kao zasebna cjelina
- Nadređeni i podređeni čvor (roditelj-dijete odnos) – svaki čvor koji se grana je nadređeni čvor i svaki čvor koji nije korijenski je nekom čvoru podređen

Kako bi navedeni elementi bili što jasniji, u nastavku je dan prikaz istih na slici 8.



Slika 8: Elementi stabla odlučivanja [rad autora prema [18]]

Razlozi velike popularnosti i česte upotrebe stabla odlučivanja su sljedeći: stablo je lako protumačiti – mogu ga razumjeti osobe bez analitičke ili matematičke podloge, odnosno, za njegovo tumačenje nije potrebno statističko znanje. Stablo omogućuje analitičarima prepoznavanje značajnijih varijabli i odnosa među varijablama. Budući da su stabla otporna na izvanredne vrijednosti (vrijednosti s velikim odstupanjima od ostatka skupa) i vrijednosti koje nedostaju, zahtijeva manje čišćenje podataka, a uz to, mogu vršiti klasifikaciju temeljem numeričkih i kategorijskih varijabli što znači da nisu potrebne promjene nad kategorijskim varijablama. Velika prednost stabla je i u tome što je to neparametarski algoritam – ne zahtjeva nikakvo podešavanje parametara, već samo skup podataka s kojima radi. Mane stabla su prekomjerno prilagođavanje (rezultira previše složenim modelima), predviđanje kontinuiranih

varijabli (stablo je namijenjeno prvenstveno za diskretne varijable, a ne kontinuirane) te inženjering značajki (ukoliko su podaci nestrukturirani ili postoje latentni čimbenici stablo nije optimalno rješenje). [18]

3.2.3.1. Nasumična šuma

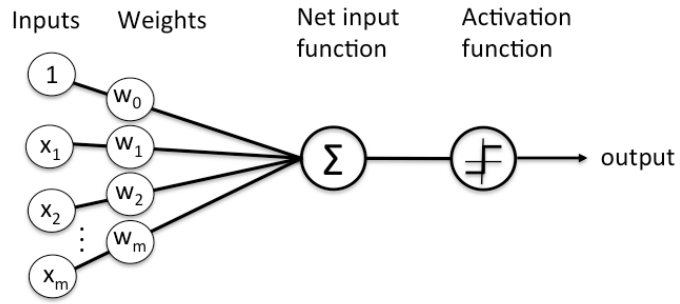
Nasumična šuma izgrađena je od mnogo stabla odlučivanja. Svako od stabla odlučivanja u nasumičnoj šumi stvoreno je pomoću podskupa atributa koji se koriste za klasifikaciju određene populacije, odnosno, svako stablo predstavlja jedno podstablo ako bi nasumičnu šumu promatrali kao obično stablo odlučivanja. Ta podstabla glasaju o tome kako klasificirati određenu instancu ulaznih podataka, a nasumična šuma razmatra te glasove kako bi odabrala najbolje predviđanje, čime se smanjuje mogućnost prekomjernog prilagođavanja. Pronalaženje korijenskog čvora i grananje vrši se nasumično. Nasumična šuma nadzirano uči kako klasificirati podatke. Kako se koristi nasumični skup podataka, algoritam kreira niz pravila koja se zatim mogu koristiti za predviđanje. [18]

3.2.4. Neuronska mreža

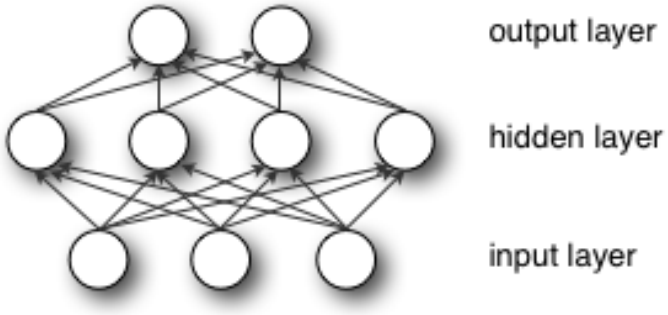
Neuronska mreža u smislu strojnog učenja računalni je sustav koji se sastoji od kolekcije povezanih jedinica – neurona koji su organizirani u slojeve. [19] Motivacija za kreiranje ovakvog modela bio je živčani sustav čovjeka. Kako bi razlikovali neuronsku mrežu živčanog sustava kod ljudi, računalna neuronska mreža naziva se još i umjetna neuronska mreža (eng. *Artificial Neural Network*, ANN). Algoritam kao takav dizajniran je da prepoznaje obrasce, odnosno da interpretira ulazne podatke putem svojevrsne percepcije te da ih označava ili grupira (klasifikacija i klasteriranje). Neuronska mreža može učiti nadzirano, nenadzirano ili polunadzirano. Obrasci koje neuronska mreža prepoznaje su numerički, sadržani u vektorima, u koje se moraju prevesti svi podaci iz stvarnog svijeta, bilo da su to slike, zvuk ili tekst. Prema području primjene, odnosno problemu koji neuronske mreže rješavaju, razvilo se nekoliko tipova neuronskih mreža, a neke od njih prema Kličeк [20] su vjerojatnosna mreža (eng. *probablistic neural netowork*, PNN), mreža učeće vektorske kvantizacije (eng. *learning vector quantization*, LQV), mreža s radijalno zasnovanom funkcijom (eng. *radial basis funciton*, RBF). Često se spominju konvolucijske neuronske mreže (eng. *convolutional neural network*, CNN) koje se koriste za rad sa slikama. Ono što se također često spominje su duboke neuronske mreže (eng. deep neural network, DNN) koje se od klasičnih neuronskih mreža razlikuju u količini skrivenih slojeva – ANN mreže imaju po jedan skriveni sloj, odnosno ukupno tri sloja, dok DNN imaju više od jednog skrivenog sloja. Što su to slojevi i koji su ostali elementi ANN objašnjeno je u nastavku.

3.2.4.1. Slojevi i neuroni

Glavni elementi neuronske mreže su neuroni, koji se često nazivaju čvorovima. Pošto je umjetna neuronska mreža kreirana prema neuronskoj mreži ljudi, i umjetni neuroni se aktiviraju prema određenoj količini podražaja. Ulaz, odnosno podražaj neurona čine ulazne vrijednosti u kombinaciji s koeficijentima ili težinama koje tu ulaznu vrijednost pojačavaju ili prigušuju. Sve ulazne vrijednosti se zbrajaju i prosljeđuju neuronu na izračun. Neuron za izračun koristi funkciju prijenosa (eng. *activation function*) kako bi odredio treba li i u kojoj mjeri taj podražaj putovati dalje kroz mrežu kako bi utjecao na krajnji ishod. Ukoliko podražaj prolazi dalje, kaže se da je neuron aktiviran. Prikaz opisanog procesa prikazan je na slici 9. Neuroni su međusobno organizirani u slojeve i povezani među slojevima (slika 10). Razlikuju se tri tipa slojeva: ulazni, skriveni i izlazni. Podaci prilikom izvršavanja algoritma učenja putuju od ulaznog kroz skriveni sloj (slojeve) do izlaznog sloja. Ulazni sloj sadrži po jedan neuron za svaku komponentu ulaznih podataka, odnosno ulaznih neurona ima onoliko koliko ima značajki (atributa) u danom skupu podataka, bez atributa koji definira oznaku, odnosno klasu u koju pripada određena instanca. Svaki od skrivenih slojeva ima proizvoljno odabran broj neurona. Izlazni sloj ima onoliko neurona koliko ima klasa rješenja. Primjer prikazan na slici 10 ima tri neurona u ulaznom sloju, četiri u skrivenom sloju i dva u izlaznom sloju. Kako ima samo jedan skriveni sloj, radi se o klasičnoj neuronskoj mreži, a ne dubokoj. Jednostavni primjer koji bi mogao biti modeliran ovakvom mrežom jest predviđanje je li osoba prekomjerne ili normalne tjelesne težine ukoliko kao ulazne varijable imamo visinu, težinu i spol. [21]



Slika 9: Prikaz procesa aktivacije neurona [21]



Slika 10: Prikaz slojeva neuronske mreže [21]

3.2.4.2. Funkcije prijenosa

U neuronskoj mreži, funkcija prijenosa je funkcija koja mapira ulaze u neuron na njegov odgovarajući izlaz. Ulaz u neuron predstavljen je kao ponderirani zbroj svake dolazne veze (sve vrijednosti inputa pomnožene odgovarajućim zadanim težinama, odnosno ponderima). Funkcija prijenosa zatim vrši matematičku operaciju koja najčešće kao rezultat daje vrijednost između neke donje i gornje granice. Ta je transformacija često nelinearna. Neke od češće korištenih funkcija prijenosa su sigmoidna funkcija i tangens hiperbolni, a popularna je i ReLu funkcija koja kao izlaz daje maksimum između nule i unesene vrijednosti – pr. ako je ulazna vrijednost u neuron -3, ReLu funckija vraća 0, a ako je ulazna vrijednost 2, vraća 2. Motivacija za kreiranje ovakvog pristupa, odnosno korištenje funkcije prijenosa koja vraća rezultat između dvije granice, je biološka aktivnost u ljudskom mozgu gdje se različiti neuroni aktiviraju različitim podražajima, npr. ukoliko osoba na dlanu drži kocku leda, receptori za hladno će se aktivirati, dok se receptori za toplo neće aktivirati. Ukoliko se aktivaciju receptora prikaže kao sigmoidna funkcija (za negativne ulaze daje rezultat jako blizu 0, za pozitivne jako blizu 1, a za vrijednosti oko nule rezultat će biti neki broj između 0 i 1), receptori za hladno imat će vrijednost bližu 1, dok će receptori za toplo imati vrijednost bliže 0. [22]

3.2.4.3. Greška neuronske mreže

Greška neuronske mreže definira se kao razlika vrijednosti između očekivane vrijednosti i rezultata, odnosno predviđanja algoritma, npr. u slučaju kada algoritam zaključuje je li na slici pas ili mačka nad označenim ulaznim slikama. Algoritam kao ulaz uzima sliku i daje predviđanje kolika je vjerojatnost za jedan izlaz – mačka, 80%, a koliko za drugi – pas, 20%. Ukoliko je na slici mačka, greška iznosi 20%. Jedna od funkcija koja se često koristi za izračun greške je srednja kvadratna pogreška (eng. *mean squared error,* MSE). MSE se izračunava na način da se najprije izračuna razlika između rezultata predviđanja i očekivane vrijednosti rezultata, a zatim se za vrijednost kvadrira. Ukoliko se algoritmu proslijedi skup podataka s više instanci, greška se računa kao prosjek svih kvadratnih greški u tom prolazu. Greška se koristi prilikom optimizacije, odnosno učenja algoritma, a ideja je da se prilikom tog procesa učenja greška smanjuje, odnosno da algoritam točnije predviđa rješenja. [23]

3.2.4.4. Treniranje, validacija i testiranje modela

Treniranje modela zapravo je rješavanje problema optimizacije modela – cilj treniranja je optimizirati sve pondere u modelu kako bi on što točnije mapirao ulazne vrijednosti s njihovim izlaznim oznakama, odnosno izlaznim klasama. Ponderi se optimiziraju odabranim algoritmom za optimizaciju, a sam postupak optimizacije samim time ovisi o odabranom algoritmu. Najpoznatiji algoritam za optimizaciju je stohastički gradijentni spust (eng. *stochastic gradient descent*, SGD). Cilj SGD algoritma je minimizirati grešku neuronske mreže. Kako bi mogao

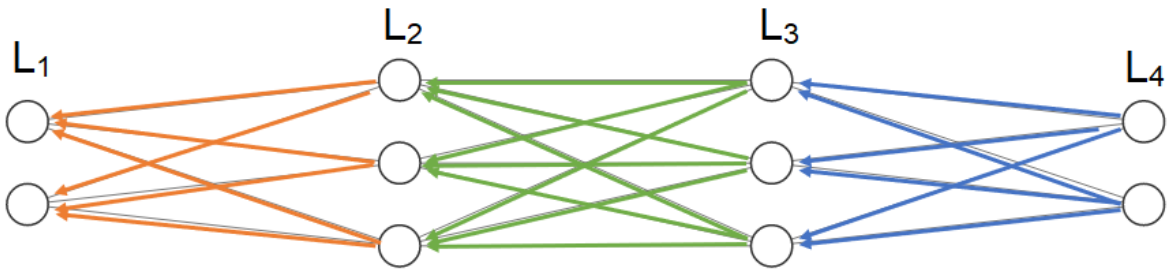
minimizirati grešku, potrebno je da podaci prođu kroz model, izračuna se greška, ponderi se optimiziraju te se podaci ponovo puštaju kroz model. Takav postupak opetovanih slanja istih podataka putem mreže smatra se treniranjem odnosno obukom modela. Jedan prolaz svih podataka iz skupa za treniranje naziva se epohom. Treniranje modela najčešće se sastoji od više epoha, a sama epoha najčešće se dijeli u serije (eng. *batch*) podataka koje se plasiraju modelu odjednom (npr. neki skup podataka ima 100.000 zapisa koji se kroz model plasiraju u serijama od 5.000 kroz 100 epoha – to znači da će jedna epoha imati 20 prolaza različitih serija podataka kroz model).

Kako prilikom primjene strojnog učenja za rješavanje određenog problema imamo dostupan samo jedan skup podataka, potrebno je iz tog skupa kreirati skup podataka za treniranje, za validaciju i za testiranje modela, pri čemu je skup za treniranje opsegom najveći. Skup za treniranje koristi se za opetovano slanje kroz mrežu kako bi model učio. Validacijski skup podataka koristi se prilikom treniranja modela kako bi se tokom samog treniranja provjerilo koliko kvalitetno je model nešto naučio, odnosno za dani ulaz iz validacijskog skupa, predviđa se rezultat te se provjerava je li rezultat dobar. Skup za testiranje specifičan je po tome što je neoznačen te su podaci iz skupa za testiranje novi, odnosno oni koje model još nije vidio. To omogućava da se provjeri stvarna točnost modela jer se prema tom ulazu model odnosi kao i prema svakom drugom ulazu kojeg bi jednog dana obrađivao i temeljem toga predviđao rezultat. [24]

3.2.4.5. Kako neuronska mreža uči

Nakon svakog prolaza podataka kroz model, izračuna se greška. Izračunata greška koristi se za izračun gradijenta (derivacija funkcije više varijabli) te greške s obzirom na svaki od pondera u mreži. Na razini jednog pondera, gradijent greške koristimo kako bi ažurirali vrijednost tog pondera. Ažuriranje pondera najčešće se provodi pravilom širenja greške unatrag (eng. *back propagation*) na sljedeći način (slika 11): pr. mreža se sastoji od četiri sloja, ulaznog, dva skrivena sloja i izlaznog, redom označeni slovima L_1, L_2, L_3, L_4. Gradijent greške računa se za pondere koji se ažuriraju jer se on izračunava u ovisnosti o nekom ponderu. Najprije se ažuriraju ponderi veza između slojeva L_3 i L_4 (plavo). Nakon toga ažuriraju se ponderi veza između slojeva L_2 i L_3 (zeleno) te se na kraju ažuriraju preostali ponderi između ulaznog i prvog skrivenog sloja (naračnasto). Na taj način se jednom izračunata greška mreže širi od posljednjeg, izlaznog sloja prema ulaznom sloju. Dano objašnjenje podosta je jednostavno, ali dovoljno dobro da se intuitivno shvati način na koji širenje greške unatrag korigira pondere. Razlog zbog kojeg se izračunava baš na takav način leži u tome što je gradijent derivacija funkcije više varijabli, a za izračun gradijenta nekog neurona koristi se izlaz njegove funkcije prijenosa, koji zapravo predstavlja ponder veze između tog neurona i onog

nakon njega, koji će u slučaju širenja greške unatrag već biti ažuriran, dok u suprotnom to ne bi bio slučaj.



Slika 11: Prikaz širenja greške unatrag [rad autora]

Prilikom same korekcije pondera važna je još jedna varijabla, a to je koeficijent učenja. Koeficijent učenja je vrijednost koja označava veličinu koraka smanjenja, odnosno ažuriranja pondera, a najčešće ima vrijednost između 0,01 i 0,001. Kako bi algoritam izračunao novu vrijednost pondera, množi gradijent za taj ponder s koeficijentom učenja te oduzima tu vrijednost od trenutne vrijednosti pondera. Ukoliko je koeficijent učenja malen, npr. 0,001, može se desiti da se ponderi neznatno ažuriraju, što utječe na brzinu učenja – može se desiti da će biti potrebno puno epoha treniranja modela kako bi se desile željene promjene nad modelom. S druge pak strane, ukoliko je koeficijent učenja veći, npr. 0,01, može se desiti da će ažuriranje pondera biti preveliko, odnosno da će ažuriranje dati lošije rezultate nego su bili prije ažuriranja.

Ažuriranje pondera u osnovi je ono što se smatra učenjem modela – model uči koje vrijednosti treba dodijeliti svakoj težini obzirom na to kako te inkrementalne promjene utječu na grešku modela. Kako se ponderi mijenjaju, mreža postaje pametnija u smislu preciznog preslikavanja ulaza na ispravan izlaz. [25]

3.2.4.6. Prekomjerno prilagođavanje i nedovoljno prilagođavanje

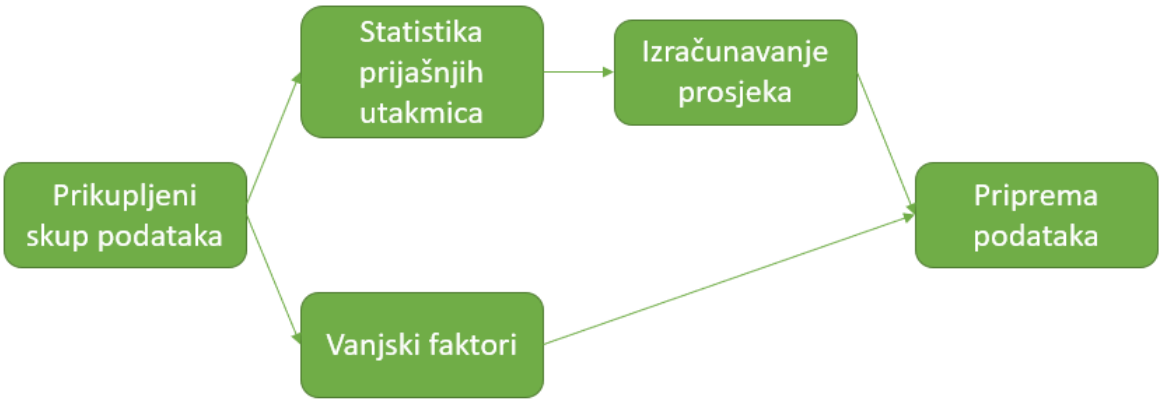
Do prekomjernog prilagođavanja dolazi kada model postane jako dobar u klasificiranju ili predviđanju za podatke koji su bili uključeni u skup podataka za treniranje modela, ali nije dobar u klasificiraju ili predviđanju za podatke koji nisu u tom skupu. Prekomjerno prilagođavanje možemo prepoznati putem metrika treniranja modela, točnosti i greške nad skupom za trening i točnosti i greške nad skupom za validaciju. Ukoliko se te vrijednosti značajno razlikuju, odnosno vrijednosti za validacijski skup su značajno lošije od onih za skup za treniranje (točnost manja, a greška veća), znači da model netočno klasificira ili predviđa rješenja za validacijski set. Koncept prekomjernog prilagođavanja svodi se na činjenicu da model nije u mogućnosti dobro generalizirati. Izuzetno je dobro naučio značajke skupa za trening, ali ako modelu damo podatke koji malo odstupaju od podataka korištenih za trening, nije u mogućnosti generalizirati i precizno predvidjeti rezultat. Tome se može doskočiti

dodavanjem još relevantnih podataka u skup za trening: npr. predviđa se je li na slici pas ili mačka, a u trenutno u skupu postoje slike samo velikih pasa, model vjerojatno neće dobro klasificirati sliku malog psa pa se u skup za trening dodaju slike malih pasa. Također, u slučaju CNN preporuča se povećanje skupa podataka odabranim transformacijama ulaznih slika poput izrezivanja slika, rotiranja, zrcaljenja, zumiranja i slično. Drugi način na koji se može spriječiti prekomjerno prilagođavanje je smanjenje kompleksnosti modela, odnosno uklanjanjem nekih slojeva ili smanjenjem neurona u slojevima. [26]

Nedovoljnim prilagođavanjem smatra se situacija kada model nije u mogućnosti dobro klasificirati podatke nad kojima je treniran. Tu situaciju prepoznajemo kada je točnost modela niska, a greška velika. Ukoliko model daje takve rezultate, vrlo vjerojatno neće dati dobre rezultate nad podacima koji su tek prvi put procesirani modelom. Kako bi se izbjeglo nedovoljno prilagođavanje, predlaže se povećati broj slojeva u modelu, povećati broj neurona te ukoliko je moguće dodati još značajki podacima. [27]

4. Strojno učenje u predviđanju sportskih rezultata

Jedno od područja koja zahtijevaju dobru točnost predviđanja je predviđanje sportskih rezultata ponajviše zbog velikih novčanih iznosa koji su uključeni u klađenje. Uz to, menadžeri i vlasnici klubova teže modelima klasifikacije kako bi mogli razumjeti i formulirati strategije potrebne za pobjedu u utakmicama. Prema Bunker, Thabath [28], dobro poznavanje domene jedan je od najvažnijih faktora prilikom predviđanja sportskih rezultata jer sami rezultati, tj. statistika vezana uz samu utakmicu ili meč najčešće nije dovoljno opsežan skup podataka. Primjerice, neki skup podataka vezan uz nogomet sadrži broj golova po utakmici, vrijeme kada su golovi zabijeni, broj žutih i crvenih kartona, posjed lopte, broj pokušaja napada na gol, broj kontra napada i slično, ali ne sadrži podatke o tome jesu li neki igrači prije utakmice pretrpjeli ozlijede, jesu li dugo putovali prije same utakmice, jesu li imali pripreme i slično. Osim konkretne metrike vezane uz samu utakmicu i vanjski faktori utječu na rezultat utakmice, a oni se najčešće ne razmatraju u samom modelu te zbog toga većina modela strojnog učenja u predviđanju sportskih rezultata postiže prosječnu točnost od 70%. Kako bi se ti rezultati poboljšali, potrebno je prikupiti više podataka koji opisuju vanjske faktore i uključiti ih u model. Nadalje, povijesni podaci utakmica trebali bi biti podvrgnuti izračunavanju prosječnih vrijednosti, tako da posljednji zapis prikazuje prosječne vrijednosti svih utakmica prije tog zapisa. Proces pripreme podataka tada izgleda kako je prikazano na slici 12. Ostali koraci procesa kreiranja modela strojnog učenja podudaraju se s klasičnim procesom kreiranja modela strojnog učenja – nakon prikupljanja podataka i odabira važnijih značajki, prema potrebi pripremiti podatke za prosljeđivanje modelu, kreirati skupove za treniranje, validaciju i testiranje, kreirati model, trenirati i zatim evaluirati rješenje.



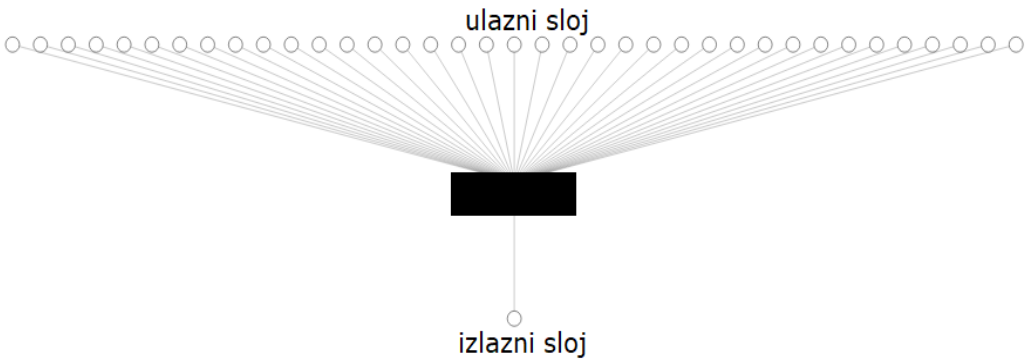
Slika 12: Proces pripreme podataka u predviđanju sportskih rezultata [rad autora prema [28]]

The Ultimate Fighting Championship (UFC) američka je mješovita borilačka vještina (eng. *mixed martial arts*, MMA). To je borbeni sport s punim kontaktom koji omogućuje

korištenje širokog spektra borbenih tehnika i vještina iz mješavine ostalih borbenih sportova. Pravila dopuštaju upotrebu tehnika udaranja i hvatanja u klinč dok borci stoje, ali i na tlu. Sudjelovanje na natjecanjima dopušteno je sportašima različitog porijekla. Stilovi borbe koji su dio MMA su boks, brazilski Jiu-Jitsu, japanski Jiu-Jitsu, Judo, Karate, kickboks, Kung Fu, slobodni stil i grčko-rimsko hrvanje i TaeKwonDo. Pobjedu u borbi moguće je ostvariti na nekoliko načina: predajom protivnika (fizička – tapkanje ili verbalna), nokautom (protivnik je onesviješten zbog udarca ili pada) te tehničkim nokautom kada sudac prekine borbu ili sudačkom odlukom koja može biti jednoglasna (svi suci odabiru jednog borca), podijeljena (jedan sudac odabere jednog, a preostala dva suca drugog borca), odluka većine (glavni sudac kaže da je neriješeno dok preostala dva suca odaberu istog borca kao pobjednika). Meč također može završiti bez pobjednika, odnosno neriješenim rezultatom. [29]

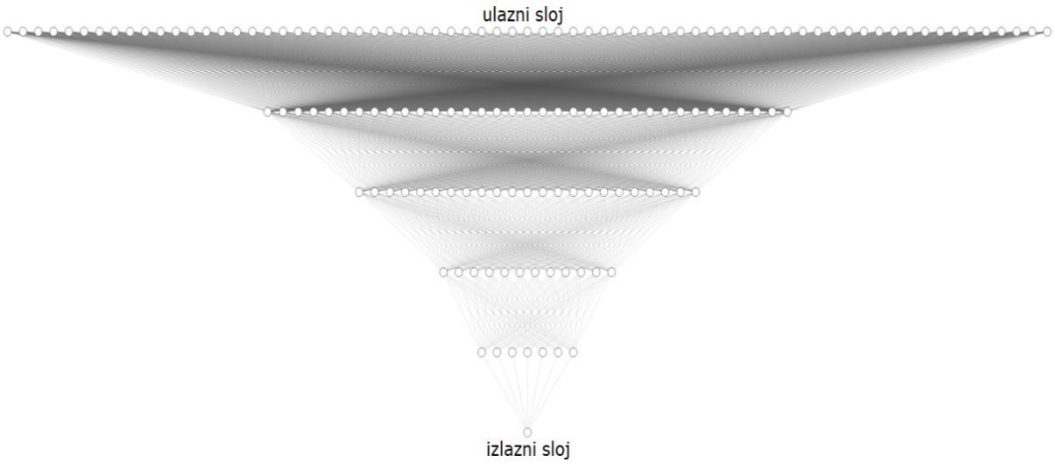
Motivacija za izradu praktičnog dijela rada leži u znatiželji u kojoj je mjeri moguće predvidjeti sportske rezultate i na koji način, odnosno, kako implementirati odabrani algoritam strojnog učenja – neuronsku mrežu. Pretpostavlja se da će model postići točnost predviđanja od oko 60%. Skup podataka koji se koristi pronađen je na internetu, a vezan je uz individualni sport, UFC borbe.

Programsko rješenje sastoji se od dva modela neuronske mreže. U jednom modelu korišten je Keras funkcionalni API, a u drugom Keras sekvencijalni API. Funkcionalni API zahtjeva pretvaranje ulaznih podataka u takozvane tenzore (eng. *tensor*), a model je kreiran na principu crne kutije – potrebno je definirati samo ulazni i izlazni sloj, dok su skriveni slojevi crna kutija (slika 13). Tenzori su višedimenzionalni nizovi (višedimenzionalna polja) ujednačenog tipa podataka. Veoma su slični poljima Python NumPy paketa. Tenzori se ne mogu mijenjati, odnosno ažurirati, već je potrebno stvoriti novi. Mogu poprimiti oblik skalara (jedna vrijednost), vektora (jednodimenzionalno polje) i matrice s više dimenzija (pr. [3,3,3] može se zamisliti kao Rubikova kocka).



Slika 13: Prikaz strukture tenzor modela [rad autora]

Sekvencijalni API omogućava kreiranje modela slaganjem linearnog niza slojeva pri čemu je potrebno definirati svaki sloj zasebno, kako je prikazano na slici 14 (broj neurona u slojevima je veći nego što je prikazano).



Slika 14: Prikaz strukture sekvencijalnog modela [rad autora]

Modeli se razlikuju u pripremi podataka, odnosno načinu normalizacije numeričkih vrijednosti i kodiranja kategorijskih varijabli te samoj izradi modela i predikciji rezultata. Kompiliranje i treniranje modela su gotovo isti. Radi boljeg razumijevanja, model u kojem je korišten Keras funkcionalni API naziva se tenzor model, dok se drugi model naziva sekvencijalni model. Tenzor model izrađen je prema članku dostupnom na službenim Keras stranicama [30], dok je sekvencijalni model izrađen prema uputama sa službenih TensorFlow stranica [31].

U nastavku su opisani koraci kreiranja rješenja uz odgovarajuće isječke programskog koda. Kako bi se kreiralo rješenje, potrebno je najprije učitati Python biblioteke odnosno pakete koji će omogućiti rad sa skupovima podataka (pandas, numpy) i neuronskom mrežom (tensorflow, keras).

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
```

Sljedeći korak u izradi modela je pripremiti odabrani skup podataka kako bi se mogao iskoristiti za treniranje, validaciju i testiranje modela.

4.1. Opis izabranog skupa podataka

Zbog velike popularnosti sporta, postoji mnogo internetskih stranica na kojima se mogu naći podaci o povijesnim rezultatima za mnoštvo različitih sportova. Prikupljanje sirovih

podataka s takvih izvora zahtijeva puno truda i vremena pa je za potrebe praktičnog dijela ovog rada pronađen skup već ranije prikupljenih podataka. Radi se o skupu koji sadrži podatke o UFC borbama od 1993. do 2019. godine koji su prikupljeni s web stranice UFC Stats (http://www.ufcstats.com/statistics/events/completed) [1]. Warrier u opisu navodi da je sirove podatke skupio korištenjem Python Beautiful Soup paketa koji služi za izvlačenje podataka iz HTML i XML datoteka koje je zatim obradio korištenjem funkcionalnosti Python Pandas paketa. Na stranici su dostupni prikupljeni sirovi podaci te obrađeni skup podataka. Za ovaj rad korišten je obrađeni skup podataka *data.csv*.

Skup podataka sastoji se od 145 atributa i 5144 zapisa. Svaki zapis (redak) sadrži statistike oba borca, pojedinostima borbe i pobjedniku. Borce predstavljaju „crveni“ i „plavi“ za crveni i plavi kut. Statistike koje su prikazane u retku odnose se na sve prijašnje borbe boraca, a prikazane su i odvojeno za prijašnje borbe trenutnih dvaju protivnika (atributi koji sadrže kraticu „opp“). Atributi vezani uz crvenog borca označeni su prefiksom „R_“ dok su atributi vezani uz plavog borca označeni prefiksom „B_“. Zavisnu varijablu predstavljat će stupac „Winner“ koji označuje tko je pobjednik, odnosno je li borba završila izjednačenim rezultatom. U nastavku su dana objašnjenja nekih atributa:

- Osnovni podaci o borcu: fighter – ime borca, age – starost, height_cms – visina, weight_lbs – težin, reach_cms – doseg borca (raspon ruku), Stance – stav u borbi, wins – broj pobjeda u karijeri, losses – broj izgubljenih borbi u karijeri, draw – broj borbi s izjednačenim rezultatom u karijeri
- Osnovni podaci o borbi: date – datum borbe, location – lokacija borbe, referee – sudac, title_bout – borba za titulu, weight_class – razred težine, no_of_rounds – broj predviđenih rundi u borbi, last_round – broj posljednje runde borbe, last_round_time – vrijeme završetka borbe, win_by – metoda pobjede
- Statistički podaci o borbama – razlikuju se dva tipa, „att“ koji označava pokušaj i „landed“ koji označava udarac: BODY – udarac u tijelo, HEAD – udarac u glavu, CLINCH – udarac u klinču, GROUND – udarac na tlu, REV – broj kontri, SUB_ATT – broj pokušaja predaje, SIG_STR – ukupan broj značajnih udaraca, TOTAL_STR – ukupan broj udaraca, TD – broj obaranja, KD – broj nokauta

Podaci su u programsko rješenje učitani korištenjem naredbe iz pandas paketa za učitavanje podataka iz .csv datoteke, učitani podaci spremaju se u pandas DataFrame objekt, a kod je prikazan u nastavku.

```
# Učitavanje podataka iz csv datoteke u DataFrame
data = pd.read_csv("data.csv")
```

U sljedećem poglavlju opisan je proces čišćenja podataka.

4.2. Čišćenje podataka

Za model strojnog učenja važno je kakve podatke koristimo za treniranje, validaciju i testiranje. Potrebno je koristiti samo značajke i vrijednosti koje doprinose rješavanju problema, dok one koje ne doprinose treba ukloniti kako ne bi nepotrebno opterećivale sam model i proces treniranja. U nastavku je opisan proces čišćenja podataka proveden nad odabranim skupom podataka s pripadajućim isječcima koda.

Prema Hill [32], UFC prije travnja 2001. nije imao strogo definirana pravila borbe, a od travnja te godine usvojili su standarde objedinjenih pravila mješovitih borilačkih vještina. Iz tog razloga iz skupa podataka potrebno je ukloniti sve zapise prije tog datuma, pošto su pravila bila drugačija, zasigurno se i rezultati, odnosno statistike tih boraca i mečeva znatno razlikuju od onih nakon tog datuma.

```
# Brisanje mečeva prije travnja 2001.
data = data[(data['date'] > '2001-04-01')]
```

Atributi s prefiksom „avg“ predstavljaju statistiku prijašnjih borbi određenog borca, a kako se u skupu podataka pojavljuju borci samo s jednom borbom, vrijednosti tih atributa za njihove zapise su nepoznati, odnosno sadrže NULL vrijednost. Postoji mogućnost da se i u nekim drugim zapisima pojavljuju nepoznate (NULL) vrijednosti koje bi mogle naškoditi izvršavanju samog algoritma, sve retke u kojima se pojavljuju takve vrijednosti potrebno je obrisati.

```
 # Brisanje redaka s praznim zapisima
data = data.dropna()
```

Cilj algoritma neuronske mreže predvidjeti je je li borac pobjednik ili nije. Svi zapisi u kojima je rezultat izjednačen (eng. *Draw*) ne pridonose rješenju, stoga se i ti zapisi uklanjaju iz skupa podataka.

```
# Brisanje redaka s rezultatom "Draw"
data = data[data['Winner'] != 'Draw']
```

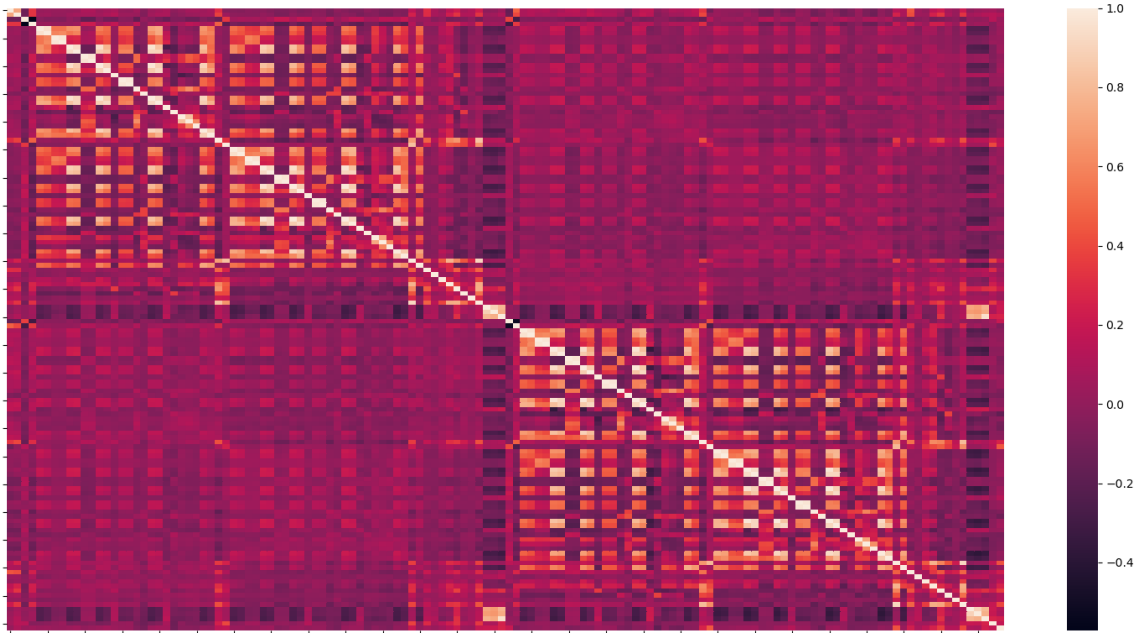
Kako je gore navedeno, izjednačene borbe ne pridonose rješenju pa se uklanjaju i stupci „R_draw“ i „B_draw“ koji opisuju u koliko se izjednačenih borbi borac borio. Stupci koji opisuju tko je bio sudac i lokaciju borbe također nisu relevantni za rješavanje problema.

```
# Brisanje atributa B_draw, R_draw, Referee, location
data.drop(['B_draw', 'R_draw', 'Referee', 'location'], axis = 1,
inplace = True)
```

Nakon svih opisanih brisanja, preostaje 3146 zapisa i 141 atribut. Matricom korelacije provjerava se kakvi su suodnosi, odnosno međusobna povezanost varijabli. Matrica korelacije prikazana je na slici 15. Za kreiranje matrice korišten je sljedeći kod:

```
import seaborn as sns
corrMatrix = data.corr()
sns.heatmap(corrMatrix, annot=False)
plt.show()
```

Iz matrice korelacije vidljivo je da podaci dvaju boraca međusobno ne koreliraju (donji lijevi i gornji desni kut). Gornji lijevi kut prikazuje plavog borca, a donji desni crvenog. Iz matrice je vidljivo da poneki atributi međusobno koreliraju, dok velik broj njih ne korelira ili slabo korelira. Detaljnijom analizom utvrđeno je da koreliraju atributi koji opisuju pokušaj zadavanja udarca i pogodak tog pokušaja, pr. „avg_BODY_att" – prosječni broj pokušaja udarca u glavu i „avg _BODY_landed“ – prosječni broj pogođenih udaraca u glavu.



Slika 15: Matrica korelacije [rad autora]

Ovakva matrica korelacije daje naslutiti da će predviđanje pobjednika biti zahtjevno, odnosno da se iz skupljenih podataka teško može odrediti tko je pobjednik meča.

4.3. Priprema podataka za modeliranje

Preuzeti skup podataka u jednom retku prikazuje podatke za dva borca, plavog i crvenog. Potrebno je odvojiti te podatke u dva zasebna skupa podataka, jedan za plavog, jedan za crvenog borca, maknuti prefikse „B_“ i „R_“ te ponovo spojiti u jedan skup podataka. U stupcu „Winner“ upisano je „Red“ ili „Blue“ što također treba promijeniti, ukoliko se radi o

skupu za plavog borca i u stupcu „Winner“ je upisano „Blue“, upisuje se 1, a u suprotnom 0. Analogno je potrebno učiniti i za crvenog borca. U nastavku je prikazan isječak koda za plavog borca.

```
# Kreiranje filtera koji uzima stupce vezane uz plavog borca
filter_blue = [col for col in data if col.startswith('B_') or col == 
'date' or col == 'Winner' or col == 'title_bout' or col == 
'weight_class' or col == 'no_of_rounds']
# Dodavanje stupaca za plavog borca u novi set podataka blue_data
blue_data = pd.DataFrame(data, columns = filter_blue)
# Preimenovanje stupaca kako bi se kasnije spojila dva seta podataka 
(red_data i blue_data)
columns_blue = []
for col in filter_blue:
    columns_blue.append(col.lstrip('B_'))
blue_data.columns = columns_blue
# U stupcu Winner za plavog borca, ako je vrijednost Blue upiše 1, 
inače 0
blue_data['Winner'] = blue_data['Winner'].replace(['Red'],0)
blue_data['Winner'] = blue_data['Winner'].replace(['Blue'],1)
```

Nakon što je isto napravljeno za crvenog borca, potrebno je spojiti te skupove podataka.

```
all_data = pd.DataFrame(red_data)
all_data.append(blue_data)
```

Nazivi nekih stupaca sadrže posebne znakove (zagrade, kosa crta) pa su preimenovani. Stupac „Winner“ je iz sredine premješten na mjesto posljednjeg stupca. Tip podatka stupca „title_bout“ je promijenjen iz tipa bool u tip int.

```
all_data = all_data.rename(columns = {'total_time_fought(seconds)': 
'total_time_fought_seconds', 'win_by_KO/TKO': 'win_by_KO_TKO'}, 
inplace = False)
winner = all_data['Winner']
all_data.drop(['Winner'], axis = 1, inplace = True)
all_data.insert(72, "Winner", winner)
all_data['title_bout'] = all_data['title_bout'].replace(True, 'True')
all_data['title_bout'] = all_data['title_bout'].replace(False, 
'False')
all_data['title_bout'] = all_data['title_bout'].replace('True', 
int(1))
all_data['title_bout'] = all_data['title_bout'].replace('False', 
int(0)
```

Modelu neuronske mreže se zasebno prosljeđuje skup ulaznih podataka, a zasebno njihove oznake, odnosno vrijednosti zavisne varijable, stoga je potrebno iste odvojiti.

```
# Odvajanje ulaznih atributa od izlaza (atribut Winner)
output_data = all_data["Winner"]
all_data.drop(['Winner'], axis = 1, inplace = True)
input_data = all_data.copy()
```

Neuronska mreža radi samo s numeričkim tipovima podataka te je potrebno sve kategorijske varijable zamijeniti numeričkim. Postoji nekoliko načina kako se to može odraditi. Najjednostavniji način je zamijeniti kategorije, pr. nazive gradova Varaždin, Zagreb, Čakovec, vrijednostima 1,2,3. Takav način nije najbolje rješenje jer će neuronska mreža zaključiti da je 1+2=3, odnosno Varaždin + Zagreb = Čakovec. Najčešće korišten način kodiranja kategorijskih varijabli je takozvano *one-hot* kodiranje. Ono radi na način da kreira n-torke koje zamjenjuju pojedinu vrijednost iz skupa vrijednosti pojedine kategorijske varijable. N-torka je duljine broja različitih vrijednosti, a sama pojedina vrijednost predstavljena je na način da na određenoj poziciji ima broj 1, dok na ostalima ima vrijednost nula, pr. Varaždin (1,0,0), Zagreb (0,1,0), Čakovec (0,0,1). U tenzor modelu to je odrađeno kroz definiranje funkcije *encode_string_categorical_feature* [30]. U sekvencijalnom modelu iskorištena je funkcija pandas paketa *get_dummies* koja ne kreira n-torke, već za svaku vrijednost pojedine kategorijske varijable dodaje novi stupac i ukoliko neki redak ima tu vrijednost varijable, u tom stupcu za taj redak je upisano 1, u suprotnom 0.

```
# Kodiranje kategorijskih varijabli
input_e = pd.get_dummies(input_data)
```

Normalizacija numeričkih varijabli pospješuje performanse modela, odnosno procesiranje modela je brže. Za normalizaciju u tenzor modelu definirana je funkcija preuzeta sa stranica Keras dokumentacije, *encode_numerical_feature* [30]. U sekvencijalnom modelu iskorištena je formula za pretvaranje svih vrijednosti nekog skupa u vrijednosti između 0 i 1, odnosno min-max normalizacija, koja najvećoj vrijednosti u stupcu daje vrijednost 1, najmanjoj 0, a preostale poprimaju vrijednosti između 0 i 1.

```
# Normalizacija numeričkih varijabli
input_e_n = (input_e - input_e.min()) / (input_e.max() - input_e.min())
```

Prije kreiranja modela, potrebno je odvojiti skupove podataka za treniranje, validaciju i testiranje. Keras model: najprije su kreirani skupovi za testiranje, validaciju i treniranje.

```
# Kreiranje skupa podataka za treniranje, validaciju i testiranje
test_data = all_data.sample(frac=0.05, random_state=1506)
all_data = all_data.drop(test_data.index)
val_data = all_data.sample(frac=0.2, random_state=1387)
train_data = all_data.drop(val_data.index)
```

Skupove je potrebno transformirati iz DataFrame objekta u tf.data.Dataset objekt (prikazan kod nad skupom za treniranje) te im odrediti veličinu serije (eng. *batch*) za obradu.

```
# Kreiranje tf.data.Dataset objekta
labels = train_data.pop("Winner")
ds = tf.data.Dataset.from_tensor_slices((dict(train_data), labels))
ds = ds.shuffle(buffer_size=len(train_data))
```

Ulaz u model potrebno je definirati kao polje tenzora, odnosno svaki atribut je potrebno pretvoriti u tenzor (prikazan primjer za jedan numerički i jedan kategorijski atribut).

```
# Numerički atributi kao Keras tenzor
title_bout = keras.Input(shape=(1,), name = 'title_bout')
# Kategorijski atributi definirani kao string kao Keras tenzor
fighter = keras.Input(shape=(1,), name = 'fighter', dtype = 'string')
```

Skriveni slojevi ovog modela rade s kodiranim i normaliziranim vrijednostima, potrebno je sve tenzore transformirati odgovarajućim funkcijama i pripremiti za prosljeđivanje modelu:

```
# Numerički atributi normalizirani
title_bout_encoded = encode_numerical_feature(title_bout,
'title_bout', ds)
# Kategorijski atributi definirani kao string one-hot kodirani
fighter_encoded = encode_string_categorical_feature(fighter,
'fighter', ds)
```

Sekvencijalni model: prema gore navedenim isječcima koda, odrađena je normalizacija numeričkih varijabli i kodiranje kategorijskih varijabli. Nakon toga odvojeni su podaci za testiranje, dok su podaci za validaciju odvojeni prilikom samog treniranja modela (`validation_split=0.2`).

```
# Kreiranje skupa podataka za treniranje i testiranje
test_input = input_e_n.sample(frac=0.1, random_state=1506)
test_output = output_data.sample(frac=0.1, random_state=1506)
# Brisanje podataka za testiranje iz skupa za treniranje na oba skupa,
input i output
input_e_n = input_e_n.drop(test_input.index)
output_data = output_data.drop(test_output.index)
```

Keras modelu se prilikom kreiranja mogu proslijediti klasična polja podataka, polja tenzora ili tf.data.dataset objekti. Zbog toga je kod sekvencijalnog modela potrebno pandas DataFrame transformirati u polje podataka.

```
# Pretvaranje pandas DataFrame-a u numpy array, x - podaci za ulaz, y
- izlaz, x_t, y_t - podaci za testiranje
x = input_e_n.to_numpy()
```

```
y = output_data.to_numpy()
x_t = test_input.to_numpy()
y_t = test_output.to_numpy()
```

Nakon iscrpne pripreme podataka kreirani su modeli što je prikazano u nastavku.

4.4. Kreiranje modela

Na slikama 13 i 14 prikazane su strukture tenzor i sekvencijalnog modela. Sekvencijalni model ima poveći broj ulaza zbog načina one-hot kodiranja korištenjem pandas `get_dummies()` funkcije. Iz tog razloga potrebno je kreirati složeniji model kako bi se dobila što veća točnost treniranog modela. Kako promjene određenih parametara modela utječu na točnost modela prikazano je u sljedećem poglavlju. U nastavku je dani kod za tenzor model.

```
# Kreiranje modela
x = layers.Dense(37, activation="tanh")(all_features)
output = layers.Dense(1, activation="tanh")(x)
model = keras.Model(all_inputs, output)
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
model.compile(opt, "binary_crossentropy", metrics=["accuracy"])
model.fit(ds, epochs=200, validation_data=vd)
```

Ovaj tip modela kao ulazne vrijednosti prima listu tenzora „all_inputs“ te izlazni sloj „output“. Vrijednosti tenzora prosljeđuju se crnoj kutiji. Sloj x definiran je kao dio te crne kutije sa 37 neurona koji primaju ulazne podatke, transformiraju ih prema danim funkcijama za normalizaciju/kodiranje, primjenjuju funkciju prijenosa tangens hiperbolni te rezultate prosljeđuju dalje kroz crnu kutiju prema izlaznom sloju „output“. Izlazni sloj sastoji se od jednog neurona jer je rješenje samo jedna klasa – pobjednik. Ukoliko je izlaz manji od 0.5, pretpostavka je da borac nije pobjednik, a ako je veći od 0.5 pretpostavka je da on jest pobjednik. Odabrani algoritam za optimizaciju, odnosno učenje mreže jest SGD s koeficijentom učenja 0.0001. Nakon kreiranja modela, potrebno ga je kompilirati korištenjem funkcije `model.compile()`. Funkcija koja računa grešku je binarna entropija, najčešće korištena funkcija za izračun greške neuronske mreže. Model je spreman za treniranje, a trenira se korištenjem funkcije `model.fit()` kojoj se prosljeđuje skup podataka za treniranje, broj epoha treniranja te skup podataka za validaciju. Sekvencijalni model kreira se na veoma sličan način:

```
# Kreiranje modela
model = Sequential([
    Dense(units=700, input_shape=(1395,), activation='tanh'),
```

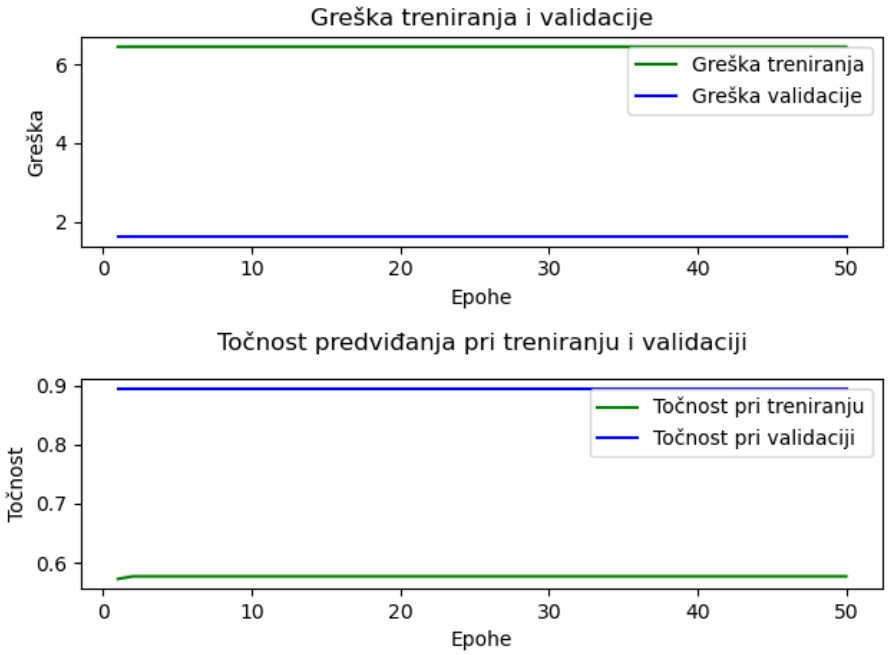
```
    Dense(units=470, activation='tanh'),
    Dense(units=250, activation='tanh'),
    Dense(units=90, activation='tanh'),
    Dense(units=1, activation="tanh")
])
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x=x, y=y, validation_split=0.2, batch_size=22, epochs=200,
shuffle=True)
```

U ovom modelu svaki sloj se definira zasebno, osim ulaznog sloja, koji je definiran putem prvog skrivenog sloja (`input_shape`). Dense označava sloj u kojem su svi neuroni povezani sa svakim neuronom u sljedećem sloju. Razlika ovog modela i tenzor modela je što se veličina serije (`batch_size`) i validacijski skup podataka mogu kreirati izravno prilikom treniranja modela. Validacijski skup definiran je kao 20% skupa za treniranje. U oba skupa se prilikom treniranja prate vrijednosti greške (eng. *loss*) i točnosti predviđanja (eng. *accuracy*). Razni scenariji na koje se naišlo prilikom treniranja modela te koje promjene su provedene kako bi se dobio model koji točnije predviđa prikazani su u nastavku.

4.5. Treniranje modela

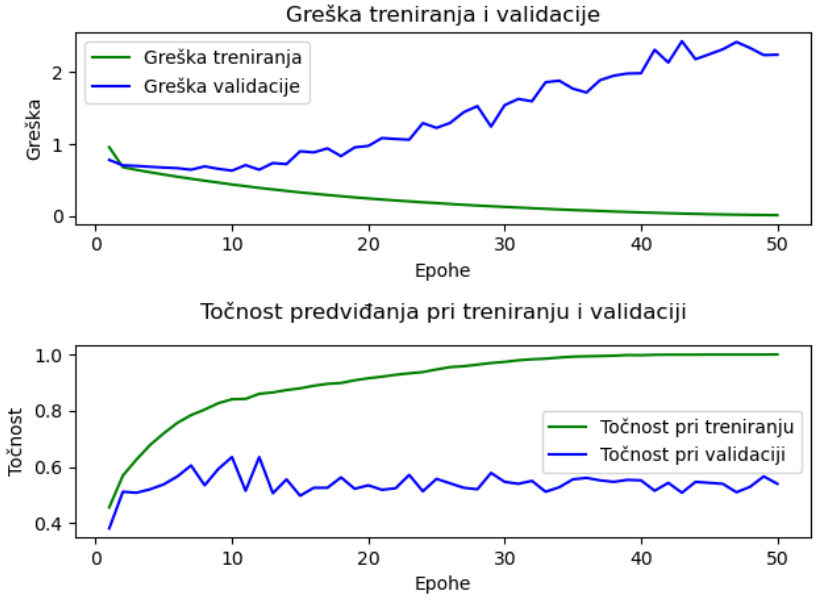
U ovom poglavlju prikazano je nekoliko scenarija treniranja sekvencijalnog modela s različitim parametrima. Za svaki scenarij prikan je grafički prikaz greške i točnosti predviđanja modela prilikom treniranja te kratak opis parametara i rezultata te ideje za promjenu parametara kako bi se model poboljšao. Ideja je prikazati tok razvoja sekvencijalnog modela. Na kraju su prikazani rezultati treniranja oba modela pri čemu su modeli definirani kako je prikazano u prošlom poglavlju. Modeli su trenirani skupom od 2265 zapisa, a validirani skupom od 566 zapisa.

Prilikom izgradnje modela savjet stručnjaka je da skriveni sloj ima broj neurona jednak aritmetičkoj sredini ulaza i izlaza. Sekvencijalni model ima 1395 ulaznih neurona, stoga skriveni sloj ima 700 neurona. Najčešće korištena funkcija optimizacije u primjerima je Adam uz koeficijent učenja 0.01, najčešće korištena funkcija prijenosa ReLu, a najčešće korištena funkcija greške binarna entropija. Rezultati treniranja prikazani su na slici 16. Iz prikaza vidimo da je greška velika (veća od 1), a točnost nešto iznad 0.6 i da model tokom treniranja uopće ne uči, što ukazuje na nedovoljnu prilagođenost (eng. *underfitting*).



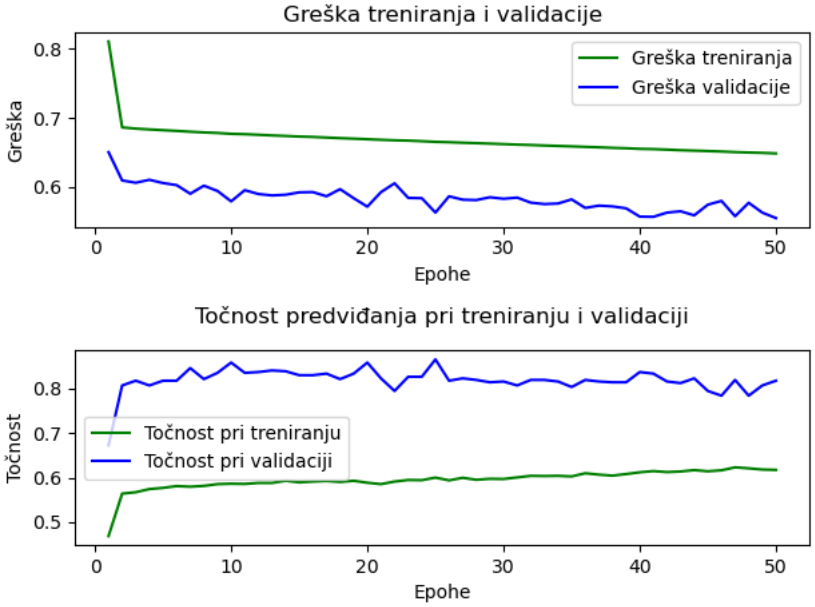
Slika 16: Grafički prikaz greške i točnosti predviđanja modela scenarij 1 [rad autora]

Koeficijent učenja promijenjen je u 0.0001. Rezultati treniranja nakon promjene prikazani su na slici 17. Iz slike vidimo da se greška modela nad podacima treniranja smanjuje i točnost povećava, ali se istovremeno vrijednosti za validacijski skup podataka kreću u suprotnom smjeru što ukazuje na prekomjernu prilagođenost modela, odnosno model dobro predviđa za podatke nad kojima trenira, a loše za sve ostale podatke. Koeficijent učenja promijenjen je u 0.001.



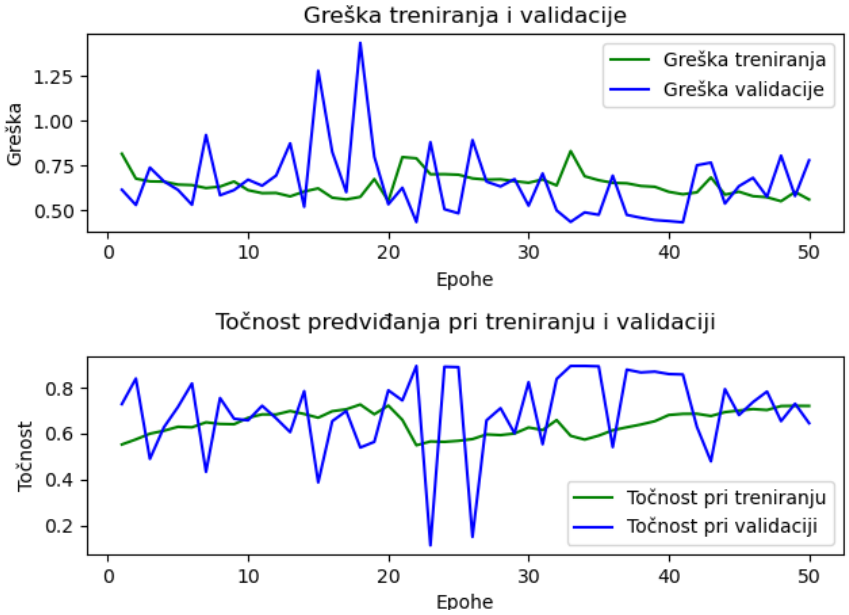
Slika 17: Grafički prikaz greške i točnosti predviđanja modela scenarij 2 [rad autora]

Tom promjenom rezultati se nisu značajno promijenili. Smanjen je broj neurona u prvom skrivenom sloju na 100. Rezultati još uvijek nisu bolji. Promijenjen je algoritam optimizacije u SGD. Rezultati su prikazani na slici 18.



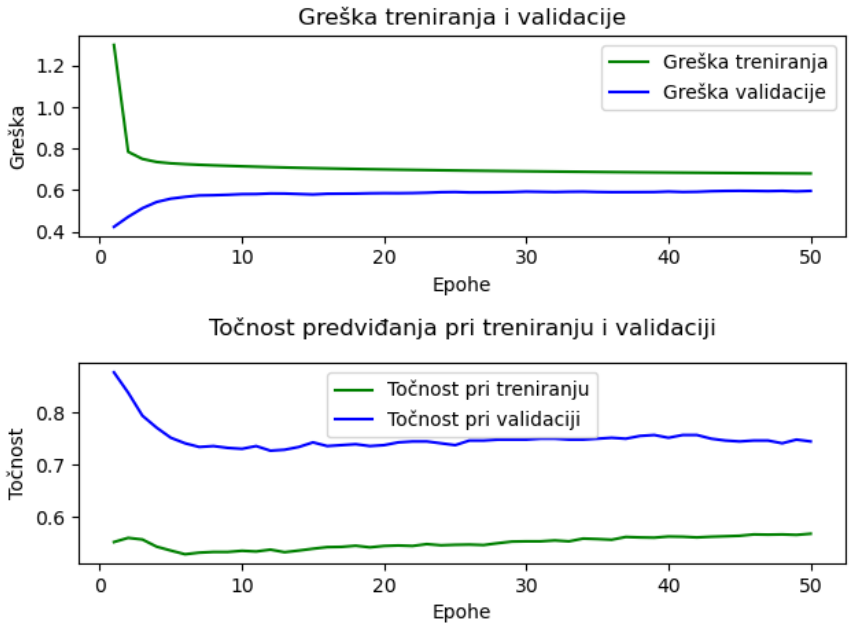
Slika 18: Grafički prikaz greške i točnosti predviđanja modela scenarij 3 [rad autora]

Iz rezultata vidljivo je da se greška tokom treniranja smanjuje, a točnost predviđanja povećava i na skupu za treniranje i na skupu za validaciju. S ciljem povećanja točnosti povećan je broj slojeva u modelu. Model je postigao malo manju grešku (0.64 na 0.62) i malo veću točnost (0.61 na 0.65). Koeficijent učenja promijenjen je na 0.01 te je umjesto ReLu funkcije prijenosa upotrijebljen tangens hiperbolni. Ova promjena iskorištena je kako bi se prikazalo koliko utjecaja ima koeficijent učenja. Rezultati su prikazani na slici 19.



Slika 19: Grafički prikaz greške i točnosti predviđanja modela scenarij 4 [rad autora]

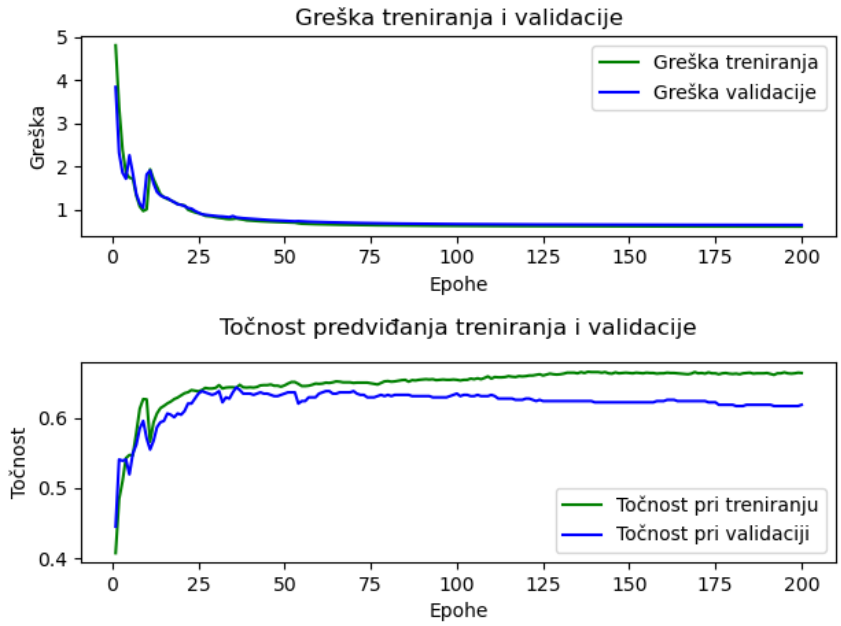
Iz slike vidimo da rezultati validacije modela skaču što je posljedica većeg koeficijenta učenja. Ukoliko se koeficijent učenja promijeni na 0.001 bez promjene ostalih parametara, više nema skokova vrijednosti (slika 20).



Slika 20: Grafički prikaz greške i točnosti predviđanja modela scenarij 5 [rad autora]

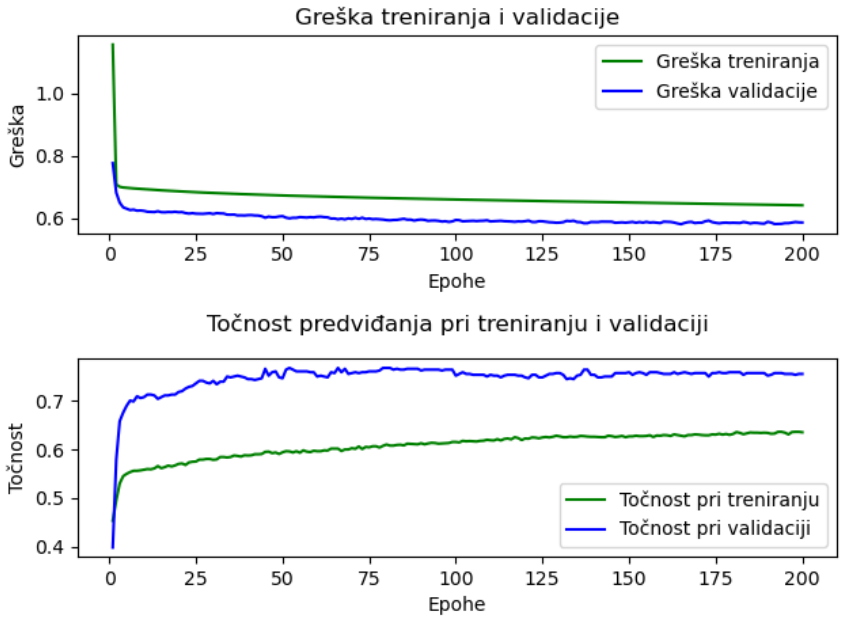
Gore navedenih nekoliko scenarija pobliže opisuje najčešće situacije koje je trebalo riješiti tokom procesa razvoja modela. Testirano je nekoliko različitih algoritama za izračun greške, optimizaciju, nekoliko kombinacija funkcija prijenosa, veličine serija te promjene koeficijenta učenja. Ni na koji način nije postignuta točnost veća od 0,65-0,70 i greška manja od 0,6-0,65, a da pritom model nije previše prilagođen. Na kraju su kreirana dva veoma slična modela čiji kod je već ranije prikazan. U nastavku su prikazani rezultati treniranja oba modela.

Na slici 21 prikazano je treniranje tenzor modela. Iz slike je vidljivo da je model u manjoj mjeri previše prilagođen, ali se greška modela na treniranju gotovo ne razlikuje od greške tokom validacije modela. Točnost predviđanja modela iznosi oko 0,65, a greška oko 0,6.



Slika 21: Grafički prikaz greške i točnosti predviđanja tenzor modela [rad autora]

Slika 22 prikazuje rezultate treniranja sekvencijalnog modela. Iz prikaza je vidljivo da model nije ni previše ni premalo prilagođen, točnost modela pri treniranju iznosi oko 0,6 kao i greška.



Slika 22: Grafički prikaz greške i točnosti predviđanja sekvencijalnog modela [rad autora]

Prema rezultatima, tenzor model točnije predviđa od sekvencijalnog modela, no u manjoj mjeri je previše prilagođen, dok sekvencijalni model daje manju točnost, no prilikom validacije dalje bolje rezultate nego tokom treniranja. U sljedećem poglavlju modeli su testirani i evaluirani.

4.6. Evaluacija modela

Evaluacija modela, odnosno testiranje posljednji je korak u procesu izgradnje rješenja, odnosno modela strojnog učenja. Koristi se kako bi se model preispitao nad podacima koje još nije vidio, odnosno nad podacima koji nisu dio ni skupa za treniranje ni skupa za validaciju. Za kreirane modele ona se vrši pozivanjem `model.evaluate()` funkcije kojoj se prosljeđuje skup testnih podataka. Modeli su evaluirani korištenjem 315 zapisa.

```
# Evaluacija modela
print('evaluation of the model')
results = model.evaluate(x_t, y_t, batch_size=22)
print('test loss, test acc:', results)
```

Evaluacija tenzor modela dala je sljedeće rezultate: greška 0.647678792476654, točnost predviđanja 0.6431095600128174. Rezultati evaluacije sekvencijalnog modela gotov se podudaraju: greška 0.6474776268005371, točnost predviđanja 0.6317460536956787. Modelima je potvrđena pretpostavka da se može postići točnost predviđanja od oko 60% te se oni mogu smatrati dovoljno dobrima za predikcije, no uvijek treba uzeti u obzir da je točnost njihovih predviđanja nešto iznad 50%, odnosno da je moguće da će pogriješiti u gotovo pola predviđanja.

4.7. Predikcije korištenjem modela

Modeli su prihvaćeni kao dovoljno dobri za predikcije rezultata, stoga je cilj ovog poglavlja modele testirati korištenjem testnih podataka i analizirati rezultate. Radi bolje preglednosti, rezultati su prikazani u tablici. Programski kod za predviđanje rezultata prikazan je u nastavku. Naredba `model.predict()` za vrijednosti iz danog skupa vraća kolika je vjerojatnost da se određeni izlaz ostvari. Modeli za predviđanje pobjednika UFC borbe definirani su jednim izlaznim neuronom, odnosno 0 ako je borac izgubio i 1 ako je pobijedio. Rezultat naredbe `model.predict()` stoga je samo jedna vjerojatnost, pri čemu vrijednosti manje od 0,5 označavaju da se pretpostavlja da će borac izgubiti, a veće od 0,5 da će pobijediti.

```
# Predikcija nad nekim vrijednostima iz testnog skupa podataka (6
instanci)
predictions = model.predict(x_t[:10])
for x in range(10):
    ind = test_ind[x]
    t_test_p = all_data_check.loc[all_data_check.index == ind]
```

```
    print(t_test_p[['fighter', 'date', 'title_bout', 'weight_class',
'Winner']])
    print("prediction:", predictions[x], "\n")
```

Prilikom predviđanja korištenjem tenzor modela potrebno je zasebno definirati vrijednost svakog atributa za instancu koju testiramo, pr. `"fighter" : t_test["fighter"]`. Zatim se sve definirane vrijednosti pretvaraju u tenzor te se `model.predict()` funkciji prosljeđuje novokreirani skup. Ukoliko se za predviđanje koristi sekvencijalni model potrebno je samo proslijediti instancu iz skupa za testiranje. Prilikom korištenja novih podataka koji se ne nalaze u testnom skupu potrebno je provesti normalizaciju i kodiranje kategorijskih varijabli na jednak način kao i za podatke kojima je model treniran.

Oba modela testirana su nad 10 istih instanci, pri čemu se sve instance zaista nalaze u testnom skupu podataka. U tablici je dano ime borca i rezultat borbe, ukoliko je u stupcu „Winner“ 1, znači da je borac bio pobjednik te borbe. Prikazane su predikcije pojedinog modela prema igraču.

Tablica 1: Prikaz rezultata predikcije [rad autora]

Testni podaci		Predikcija	
Fighter	Winner	Tenzor model	Sekvencijalni model
Gegard Mousasi	0	0.80618197	0.58318913
Rory MacDonald	1	0.7572969	0.6929079
Jeremy Stephens	0	0.4173567	0.27047867
Minotauro Nogueira	1	0.46508664	0.42732906
Jake Collier	1	0.53872395	0.5415932
John Makdessi	1	0.64150095	0.4884377
Cole Miller	1	0.62157613	0.5764956
Chris Tuchscherer	1	0.2936397	0.32550514
Nate Quarry	1	0.6058365	0.48338023
Nik Lentz	0	0.53747324	0.49465638

Ukoliko je pozadina ćelije zelena, predikcija je točna, u suprotnom je netočna. Tenzor model dao je šest točnih predikcija (60%), a sekvencijalni model pet (50%). Može se zaključiti kako oba modela očekivano predviđaju rezultate, iako tenzor model prema rezultatima treniranja i validacije pokazuje manju pretreniranost, odnosno prekomjernu prilagođenost. Iz predikcija također vidimo da se iznos predikcije daleko razlikuje od točnog rezultata, pr. za borca Gegard Mousasi koji je izgubio borbu, predviđanja su da je pobijedio pri čemu iznos nije blizu 0,5 već nešto više. To bi moglo značiti da je Gegard Mousasi dobar borac koji je imao loš dan i izgubio meč, iako u većini mečeva pobjeđuje. Suprotnost je borac Chris Tuchscherer koji je meč pobijedio, a predviđanja su relativno blizu nuli. Iz toga se može zaključiti da je taj borac većinu svojih mečeva izgubio, no taj jedan je ipak pobijedio. Takvi rezultati potvrđuju činjenicu

da je UFC veoma nepredvidiv sport i da je veoma zahtjevan i izazovan zadatak predvidjeti točne rezultate.

4.8. Usporedba modela

U ovom poglavlju dana je usporedba dvaju kreiranih modela. Modelima su zajednički svi parametri za koje je to omogućeno obzirom na samu arhitekturu, odnosno način izrade i rezultate pripreme podataka (tenzor model ima 72 ulazna neurona, dok sekvencijalni ima 1395 zbog *one-hot* kodiranja). Parametri koji se podudaraju su sljedeći:

- Funkcija prijenosa neurona – tangens hiperbolni (tanh)
- Algoritam za optimizaciju težina – stohastički gradijentni spust (SGD)
- Algoritam za izračun greške – binarna entropija (binary_crossentropy)
- Koeficijent učenja – 0,001
- Veličina serije – 22
- Broj epoha treniranja – 200
- Veličine skupova podataka: skup za treniranje 2256 zapisa, skup za validaciju 566 zapisa, skup za testiranje 315 zapisa

Tokom izrade rješenja za predviđanje rezultata UFC borbe najprije je kreiran tenzor model. Keras funkcionalni API jednostavan je za korištenje, a primjeri koji mogu poslužiti kao nit vodilja dostupni su na službenim stranicama Keras dokumentacije. Iako se svaki atribut iz ulaznog skupa podataka mora zasebno definirati kao tenzor i nakon toga, svaki tenzor jednom linijom koda normalizirati/kodirati, sama izrada modela za treniranje puno je jednostavnija nego kod sekvencijalnog modela. Olakotna okolnost je i to što su funkcije za normalizaciju i kodiranje varijabli dostupne u istoj dokumentaciji te iste ne uzrokuju povećanje broja atributa. Za sam model potrebno je definirati samo ulazni i izlazni sloj pri čemu izlazni sloj daje najbolje rezultate s brojem neurona koji je predstavljen aritmetičkom sredinom ulaza i izlaza. Osim što ne uzrokuje povećanje atributa, prednost ovog modela je što je znatno brži. Jedna epoha treniranja ovog modela traje 3 milisekunde. Prilikom evaluacije oba modela daju podjednake rezultate, no prilikom predviđanja provedenog u prethodnom poglavlju, tenzor model je ipak dao bolje rezultate.

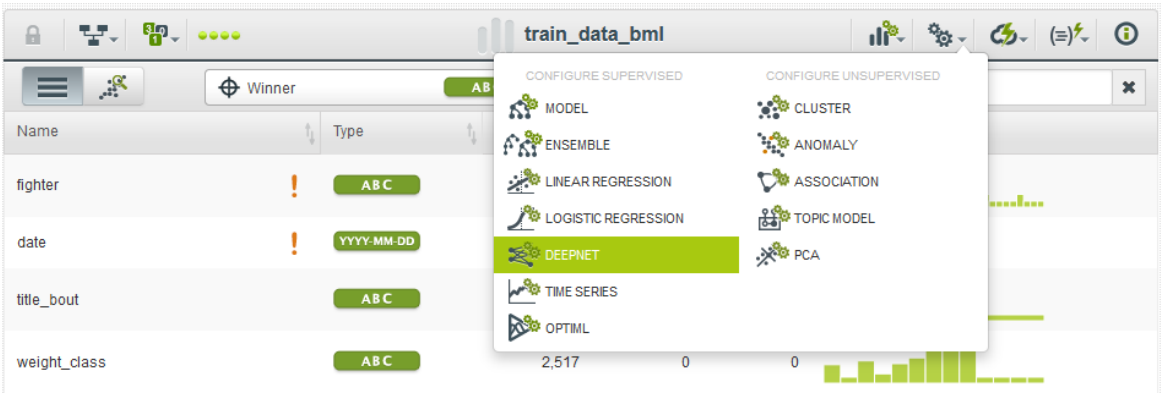
Nakon kreiranja tenzor modela postavljalo se pitanje koliko je zahtjevno kreirati model neuronske mreže u kojem je potrebno definirati sve skrivene slojeve. Jedan od većih izazova kreiranja ovog modela bilo je *one-hot* kodiranje kategorijskih varijabli. Većina dostupnih primjera koja rješava taj problem objašnjena je na manjem vrlo jednostavnom skupu podataka, koji najčešće nije sadržavao numeričke varijable. Dodatno, normalizacija podataka bila je na

primjerima objašnjena na veoma sličan način, korištenjem manjeg skupa podataka koji sadrži samo numeričke varijable. Sam proces zahtijevao je testiranje ponašanja različitih rješenja na manjem kombiniranom skupu podataka. Neka rješenja zahtijevala su da se radi nad poljem i da se posebno odvoje kategorijske varijable, kodiraju i zatim je problem bio spojiti ih ponovo s odgovarajućim numeričkim varijablama te se pritom javljalo pitanje u kojem trenutku procesa provesti normalizaciju numeričkih varijabli. Konačno rješenje je jednostavno (prikazano u poglavlju 4.3) u odnosu na proces kojim se došlo do tog rješenja. Novi problem koji je stvorilo to rješenje bio je velik broj ulaznih varijabli, skok sa 72 ulazna atributa na 1395, jer se za svaku pojedinu vrijednost kategorijske varijable kreira novi stupac. Prilikom kreiranja slojeva u modelu bilo je potrebno otkriti najbolju kombinaciju broja slojeva i broja neurona u njima. U literaturi nije pronađen nikakav konkretniji savjet kako to učiniti, odnosno koliko bi neurona trebalo biti u skrivenim slojevima i koliko skrivenih slojeva bi trebalo definirati. Cijeli proces kreiranja modela temeljio se na metodi pokušaja i pogreške, odnosno mijenjanja parametara i treniranja modela dok se nisu postigli rezultati slični tenzor modelu. Treniranje modela je zbog većeg broja neurona u mreži sporije nego kod tenzor modela. Vrijeme treniranja jedne epohe traje nešto više od jedne sekunde (1s 7ms). Sekvencijalni model pri predviđanju rezultata daje nešto lošije rezultate od predviđenog s točnošću predviđanja od 50%, no skup nad kojim se predviđalo je relativno malen stoga se razlika može zanemariti.

U online alatu za izradu modela strojnog učenja BigML kreirana je neuronska mreža. Cilj je provjeriti koju točnost predviđanja je moguće postići nad danim skupom podataka. Korišten je skup podataka dobiven nakon čišćenja i pripreme, a prije normalizacije numeričkih i kodiranja kategorijskih varijabli. Taj skup podataka podijeljen je na 2 skupa, jedan za treniranje i jedan za validaciju, pri čemu je skup za validaciju kreiran kao 20% zapisa iz skupa za treniranje. Skupovi su kreirani kao dio Pythong programskog rješenja, a kod je prikazan u nastavku.

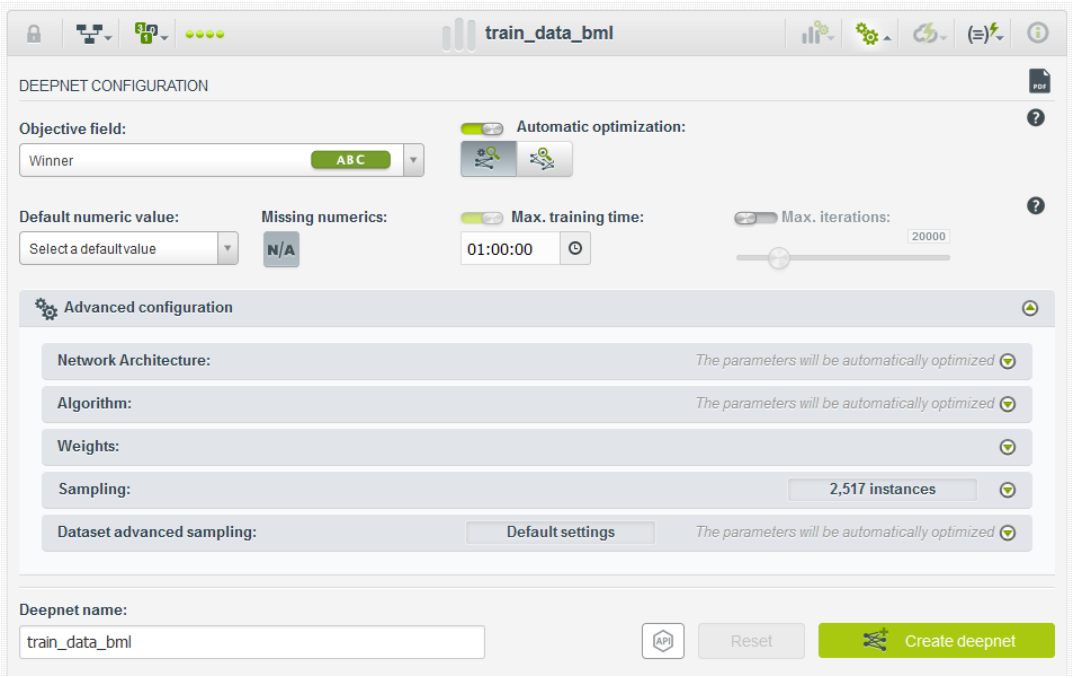
```
train_data_bml = all_data.copy()
val_data_bml = train_data_bml.sample(frac=0.2, random_state=1304)
train_data_bml = train_data_bml.drop(val_data_bml.index)
train_data_bml.to_csv('train_data_bml.csv', index_col = False)
val_data_bml.to_csv('val_data_bml.csv', index_col = False)
```

Prikaz skupa podataka te izgleda BigML alata te kako se odabere neuronska mreža prikazan je na slici 23. Nakon što je odabran *Deepnet*, odnosno neuronska mreža kao algoritam strojnog učenja za koji se kreira model, otvara se prikaz za definiranje željenih postavki prikazan na slici 24.



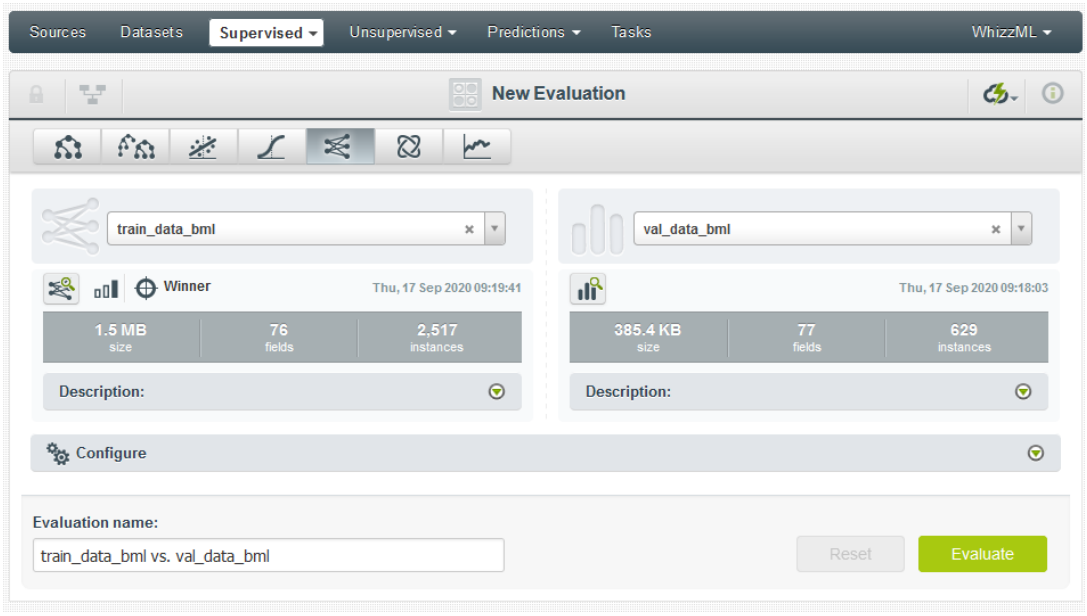
Slika 23: Prikaz skupa podataka u BigML alatu [rad autora]

Kako je cilj provjeriti u kojoj mjeri automatizirani alat bolje predviđa od samostalnog kreiranja modela, postavke se ne mijenjaju. Pritiskom na gumb *Create deepnet* kreira se neuronska mreža. Pretpostavka je da će BigML model imati nešto višu točnost predviđanja. Model se trenira skupom od 2517 zapisa.



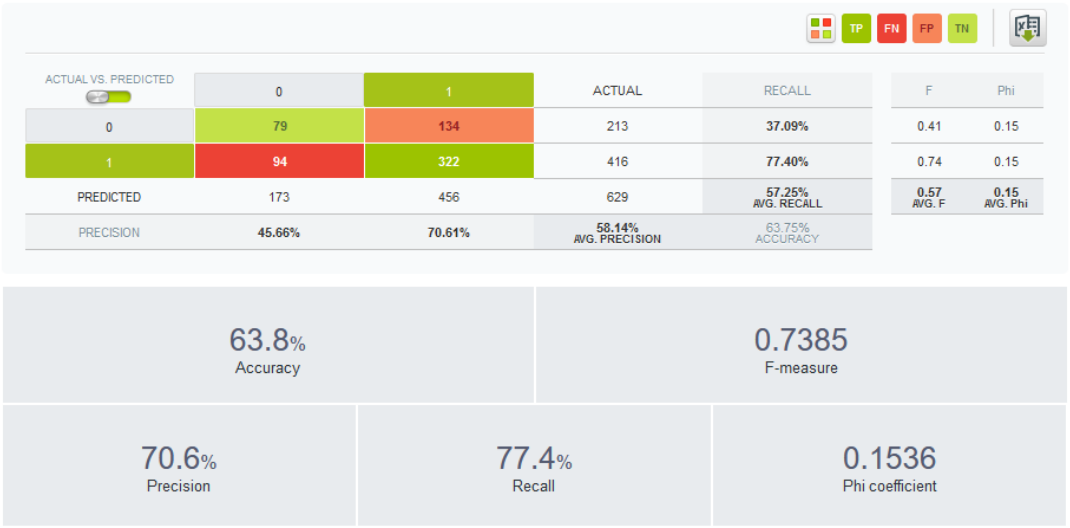
Slika 24: Postavke za kreiranje neuronske mreže u alatu BigML [rad autora]

U alatu BigML postoji opcija evaluacije modela, a prikaz podešavanja varijabli prikazan je na slici 25. Evaluira se gore definiranim validacijskim skupom podataka od 629 zapisa. Skup za treniranje ima 76 atributa, dok skup za evaluaciju ima 77 jer sadrži i ciljanu vrijednost koja se koristi za provjeru predviđanja. Oba skupa podataka u BigML alatu imaju nekoliko atributa više nego skupovi u Python rješenju jer atribut datum BigML samostalno rascijepa na vrijednosti za dan, mjesec i godinu.



Slika 25: Podešavanje evaluacije modela u BigML alatu [rad autora]

Rezultati evaluacije modela prikazani su na slici 26. Iz rezultata evaluacije vidljivo je da model postiže točnost predviđanja od 63,8% s greškom od 0,7385, što su rezultati vrlo slični onima dobivenim tenzor i sekvencijalnim modelom. Nadalje, iz tablice je vidljivo da model točnije predviđa pobjednike, čak 70,61% točnih predviđanja, dok nešto lošije predviđa kada je borac izgubio borbu, s preciznošću od 45,66%.



Slika 26: Rezultati evaluacije modela u BigML alatu [rad autora]

Prema dobivenim rezultatima može se zaključiti da su modeli kreirani u Pythonu dovoljno dobra rješenja za predviđanje pobjednika UFC borbe nad danim podacima ukoliko se uspoređuje točnost predviđanja sva tri modela.

5. Zaključak

Strojno učenje svakim danom postaje sve popularnije, a moguće ga je primijeniti u gotovo svim aspektima ljudskog života. Iako neki u tome vide prijetnju, u smislu da će jednog dana računala zamijeniti ljude, mnogo je onih koji žele na različite načine iskoristiti prednosti razvoja tehnologije i ostvariti neku korist. U ovom radu ispitano je u kojoj mjeri i na koji način se primjenom strojnog učenja mogu predvidjeti sportski rezultati.

U prvom dijelu objašnjeni su teorijski koncepti vezani uz strojno učenje – što je to strojno učenje, kako i gdje se može primijeniti, objašnjene su metode i neki od algoritama strojnog učenja s naglaskom na neuronske mreže. Spomenute su teze vezane uz modele strojnog učenja za predviđanje sportskih rezultata te smjernice za kreiranje takvog modela. U praktičnom dijelu prikazana je izrada dva modela neuronskih mreža u programskom jeziku Python korištenjem TensorFlow platforme otvorenog koda za strojno učenje i Keras sučelja za izradu neuronskih mreža. Prikaz izrade slijedi korake u procesu kreiranja modela strojnog učenja, odnosno odabir/prikupljanje podataka, čišćenje i priprema podataka, kreiranje modela, treniranje modela, evaluacija modela te predviđanje. Na kraju je nad istim skupom podataka korištenjem online alata BigML automatizacijom kreiran model kako bi se provjerilo u kojoj mjeri se preklapaju točnosti predviđanja modela kreiranih od nule, odnosno razvijenog rješenja i automatiziranog rješenja postojećeg alata.

U uvodu je, prema analizi Bunker, Susnjak [2], dana pretpostavka da će model strojnog učenja za predviđanje rezultata UFC borbi dosegnuti točnost predviđanja od 60%. Rezultati evaluacije sva tri modela daju točnost predviđanja od 64,3%, 63,1% i 63,8%, pri čemu najveću točnost predviđanja daje tzv. tenzor model. Takvi rezultati potvrđuju da je pretpostavka ispravna, a modeli se kao takvi mogu prihvatiti kao relativno dobri u predviđanju sportskih rezultata. Unapređenje modela moguće je prikupljanjem kvalitetnijeg skupa podataka, odnosno dodavanjem vanjskih faktora vezanih uz borce postojećem skupu podataka. Na taj način prema Bunker, Thabath [28] može se dobiti model s većom točnošću predviđanja.

Ovaj rad detaljno opisuje proces kreiranja neuronske mreže od nule u programskom jeziku Python na dva različita modela s prikazima isječaka koda te objašnjenjima istih. Sam proces i programski kod mogu se prilagoditi i primijeniti i na druge tipove problema, odnosno na druge skupove podataka, što se smatra najvećim doprinosom za čitatelja ovog rada.

Popis literature

[1] J. Milton, „History of sports betting“, 2017. [Na internetu]. Dostupno: https://www.bigonsports.com/history-of-sports-betting/ [Pristupano 14.9.2020]

[2] R. Bunker, T. Susnjak. *The Application of Machine Learning Techniques for Predicting Results in Team Sport: A Review*, 2019. Cornell University

[3] R. Warrier, „UFC-Fight historical data from 1993 to 2019“, 2019. [Na internetu]. Dostupno: https://www.kaggle.com/rajeevw/ufcdata?fbclid=IwAR0nJDrX029IusShg0hHVMUra9neNW6zxH16ID9J8lsH64ayA39i9CNG504#data.csv [Pristupano: 30.12.2019]

[4] Conda, 2020. [Na internetu]. Dostupno: https://conda.io/en/latest/ [Pristupano: 24.8.2020]

[5] TensorFlow (verzija 2.3.0), 2020. [Na internetu]. Dostupno: https://www.tensorflow.org/ [Pristupano: 24.8.2020]

[6] Keras (verzija 2.3.0), 2020. [Na internetu]. Dostupno: https://en.wikipedia.org/wiki/Keras [Pristupano: 24.8.2020]

[7] Pandas (verzija 1.1.2), 2020. [Na internetu] Dostupno: https://pandas.pydata.org/ [Pristupano 24.8.2020]

[8] V. Sharma, „Data Visualization for Deep Learning Model Using Matplotlib“, 2019. [Na internetu]. Dostupno: https://www.pluralsight.com/guides/data-visualization-deep-learning-model-using-matplotlib [Pristupano 14.9.2020]

[9] NN-SVG, 2020. [Na internetu]. Dostupno: http://alexlenail.me/NN-SVG/index.html [Pristupano 14.9.2020]

[10] BigML, 2020. [Na internetu]. Dostupno: https://bigml.com/ [Pristupano 14.9.2020]

[11] „Artificial Intelligence (AI) vs. Machine Learning vs. Deep Learning“. [Na internetu]. Dostupno: https://wiki.pathmind.com/ai-vs-machine-learning-vs-deep-learning [Pristupano 24.8.2020]

[12] M. Mohri, A. Rostamizadeh, A. Talwalkar, *Foundations of Machine Learning*. London: The MIT Press, 2018.

[13] Y. Kinha, „An easy guide to choose the right Machine Learning algorithm“, 2020. [Na internetu]. Dostupno: https://www.kdnuggets.com/2020/05/guide-choose-right-machine-learning-algorithm.html [Pristupano 24.8.2020]

[14] B. Kliček, „Predavanje o strojnom učenju“, nastavni materijali na predmetu Inteligentni sustavi [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2020.

[15] H. Li, „Which machine learning algorithm should I use?“, 2017. [Na internetu]. Dostupno: https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/ [Pristupano 24.8.2020]

[16] S. Ray, „Commonly used Machine Learning Algorithms (with Python and R Codes)“, 2017. [Na internetu]. Dostupno: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ [Pristupano 24.8.2020]

[17] „A Beginner's Guide to Logistic Regression For Machine Learning“. [Na internetu]. Dostupno: https://wiki.pathmind.com/logistic-regression [Pristupano 24.8.2020]

[18] „Decision Tree“. [Na internetu]. Dostupno: https://wiki.pathmind.com/decision-tree [Pristupano 24.8.2020]

[19] „Machine Learning & Deep Learning Fundamentals“. [Na internetu]. Dostupno: https://deeplizard.com/learn/video/hfK_dvC-avg [Pristupano 24.8.2020]

[20] B. Kliček, „Strojno učenje“, nastavni materijali na predmetu Inteligentni sustavi [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2020.

[21] „A Beginner's Guide to Neural Networks and Deep Learning“. [Na internetu]. Dostupno: https://wiki.pathmind.com/neural-network [Pristupano 24.8.2020]

[22] „Activation Functions In A Neural Network Explained“. [Na internetu]. Dostupno: https://deeplizard.com/learn/video/m0pIlLfpXWE [Pristupano 24.8.2020]

[23] „Loss In A Neural Network Explained“. [Na internetu]. Dostupno: https://deeplizard.com/learn/video/Skc8nqJirJg [Pristupano 24.8.2020]

[24] „Train, Test, & Validation Sets Explained“. [Na internetu]. Dostupno: https://deeplizard.com/learn/video/Zi-0rlM4RDs [Pristupano 24.8.2020]

[25] „How A Neural Network Learns Explained“ [Na internetu]. Dostupno: https://deeplizard.com/learn/video/_N5kpSMDf4o [Pristupano 24.8.2020]

[26] „Overfitting In A Neural Network Explained“. [Na internetu]. Dostupno: https://deeplizard.com/learn/video/DEMmkFC6IGM [Pristupano 24.8.2020]

[27] „Underfitting In A Neural Network Explained“ [Na internetu]. Dostupno: https://deeplizard.com/learn/video/0h8lAm5Ki5g [Pristupano 24.8.2020]

[28] R. Bunker, F. Thabath, „A machine learning framework for sport result prediction“, 2019. [Na internetu]. Dostupno: https://www.sciencedirect.com/science/article/pii/S2210832717301485 [Pristupano 14.9.2020]

[29] „About the UFC – the sport“. [Na internetu]. Dostupno: https://www.ufc.com/about/sport [Pristupano 14.9.2020]

[30] F. Chollet, „Structured data classification from scratch“, 2020. [Na internetu]. Dostupno: https://keras.io/examples/structured_data/structured_data_classification_from_scratcs/ [Pristupano 14.9.2020]

[31] „tf.keras.Sequential“, 2020. [Na internetu]. Dostupno: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential [Pristupano 14.9.2020]

[32] A. Hill, „A Timeline of UFC Rules: From No-Holds-Barred to Highly Regulated“, 2013. [Na internetu]. Dostupno: https://bleacherreport.com/articles/1614213-a-timeline-of-ufc-rules-from-no-holds-barred-to-highly-regulated [Pristupano 14.9.2020]

Popis slika

Popis slika treba biti izrađen po uzoru na indeksirani sadržaj, te upućivati na broj stranice na kojoj se slika može pronaći.

Popis tablica

Popis tablica treba biti izrađen po uzoru na indeksirani sadržaj, te upućivati na broj stranice na kojoj se tablica može pronaći.