SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

NASLOV DIPLOMSKOG RADA

Dora Katić

Split, srpanj 2021.

SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Diplomski studij: **Računarstvo**

Smjer/Usmjerenje: **Računarstvo**

Oznaka programa: 250

Akademska godina: 2020./2021.

Ime i prezime: **Dora Katić**

Broj indeksa: 512-2019

**ZADATAK DIPLOMSKOG RADA**

Naslov:

Zadatak: Napraviti pregled različitih područja primjene strojnog učenja (ML, Machine Learning), obraditi osnovne korake u strojnom učenju, od odabira i pripreme podataka do interpretacije dobivenih rezultata. Usporediti gotova programska okruženja za aplikacije strojnog učenja i napraviti pregled dostupnih javnih skupova podataka. Odabrati otvoreni skup podataka, gotovo okruženje te zadatak koji će se riješiti jednim od pristupa strojnog učenja. Prezentirati dobivene rezultate.

Prijava rada: 23.02.2021.

Rok za predaju rada:

Rad predan:

Predsjednik

Odbora za diplomski rad: Mentor:

prof. dr. sc. Maja Štula

**IZJAVA**

Ovom izjavom potvrđujem da sam diplomski rad s naslovom \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ pod mentorstvom prof. dr. sc. Maje Štule pisala samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada, te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući navela u diplomskom radu citirala sam i povezala s korištenim bibliografskim jedinicama.

Studentica

Dora Katić

**SADRŽAJ**

[1. UVOD 1](#_Toc76139315)

[2. STROJNO UČENJE 2](#_Toc76139316)

[2.1 Umjetna inteligencija 2](#_Toc76139317)

[2.2 Strojno učenje 3](#_Toc76139318)

[2.3 Odabir modela i funkcije za strojno učenje 3](#_Toc76139319)

[2.4 Nadzirano učenje 7](#_Toc76139320)

[2.5 Nenadzirano učenje 8](#_Toc76139321)

[2.6 Polu-nadzirano učenje 9](#_Toc76139322)

[2.7 Podržano učenje 9](#_Toc76139323)

[3. KLASIFIKACIJA 11](#_Toc76139324)

[3.1 Binarna klasifikacija 11](#_Toc76139325)

[3.1.1 Točnost 12](#_Toc76139326)

[3.1.2 Preciznost 12](#_Toc76139327)

[3.1.3 Odziv 13](#_Toc76139328)

[3.2 Multiclass klasifikacija 13](#_Toc76139329)

[3.2.1 One-vs-rest 14](#_Toc76139330)

[3.2.2 One-vs-one 15](#_Toc76139331)

[4. LOGISTIČKA REGRESIJA 17](#_Toc76139332)

[4.1 Gradijentni spust 19](#_Toc76139333)

[4.2 Multiclass logistička regresija 21](#_Toc76139334)

[5. NEURONSKE MREŽE 23](#_Toc76139335)

[5.1 Osnovni pojmovi 23](#_Toc76139336)

[5.2 Backpropagation algoritam 26](#_Toc76139337)

[6. MULTICLASS KLASIFIKACIJA CYBER-NAPADA 28](#_Toc76139338)

[6.1 Korištene biblioteke 28](#_Toc76139339)

[6.2 Analiza podataka 28](#_Toc76139340)

[6.2.1 Vizualizacija 32](#_Toc76139341)

[6.3 Preprocessing i feature engineering 39](#_Toc76139342)

[6.3.1 Normalizacija 40](#_Toc76139343)

[6.3.2 Label-encoding i One-hot-encoding 42](#_Toc76139344)

[6.4 Stvaranje i treniranje modela 44](#_Toc76139345)

[6.4.1 Logistička regresija 45](#_Toc76139346)

[6.4.2 Neuronska mreža 47](#_Toc76139347)

[6.5 Usporedba modela 50](#_Toc76139348)

[7. ZAKLJUČAK 51](#_Toc76139349)

[LITERATURA 52](#_Toc76139350)

[POPIS OZNAKA I KRATICA 54](#_Toc76139351)

[SAŽETAK 1](#_Toc76139352)

[KLJUČNE RIJEČI 1](#_Toc76139353)

[SUMMARY 2](#_Toc76139354)

[TITLE 2](#_Toc76139355)

[KEYWORDS 2](#_Toc76139356)

# 

# UVOD

Svakim danom sve više raste količina podataka koji se koriste za analize i rad u različitim područjima ljudske aktivnosti. Dolazi do ubrzanog tehnološkog razvoja te se događa da potrebe tehnologije nadilaze mogućnost čovjeka da samostalno proučava i pohranjuje podatke zbog čega dolazi do razvoja strojnog učenja. Strojno učenje postaje središnja tehnologija današnjice i sve je manje područja u kojima još nije došlo do njegove primjene. Ono omogućava da iz skupova podataka koji su nam na raspolaganju izvučemo novo znanje, odnosno da računalo samostalno uči iz iskustva i stvara nove spoznaje.

U ovom diplomskom radu opisani su postupci potrebni da bi se strojno učenje provelo nad skupom podataka. Opisane su glavne vrste strojnog učenja te najčešće korištene metode i pristupi za samostalno prepoznavanje uzoraka i poveznica među podacima.

U drugom poglavlju, prvenstveno je opisana umjetna inteligencija kao temelj strojnog učenja i njemu srodnih područja. Zatim je opisano samo strojno učenje kroz njegove glavne karakteristike, vrste, metode i funkcije.

Treće poglavlje opisuje klasifikaciju kao jednu od dvije glavne vrste nadziranog učenja. Predstavljena su i dva načina za njezino provođenje, odnosno binarna i višeklasna (engl. *multiclass*) metoda.

U četvrtom poglavlju opisana je logistička regresija, jedan od najčešće korištenih klasifikacijskih algoritama koji je korišten u praktičnom dijelu ovog rada za *multiclass* klasifikaciju danih podataka.

U petom poglavlju opisani su glavni pojmovi vezani za umjetne neuronske mreže koje su danas jedan od najpoznatijih i najkorištenijih algoritama strojnog učenja. Opisana je njihova osnovna struktura te model i potrebni slojevi.

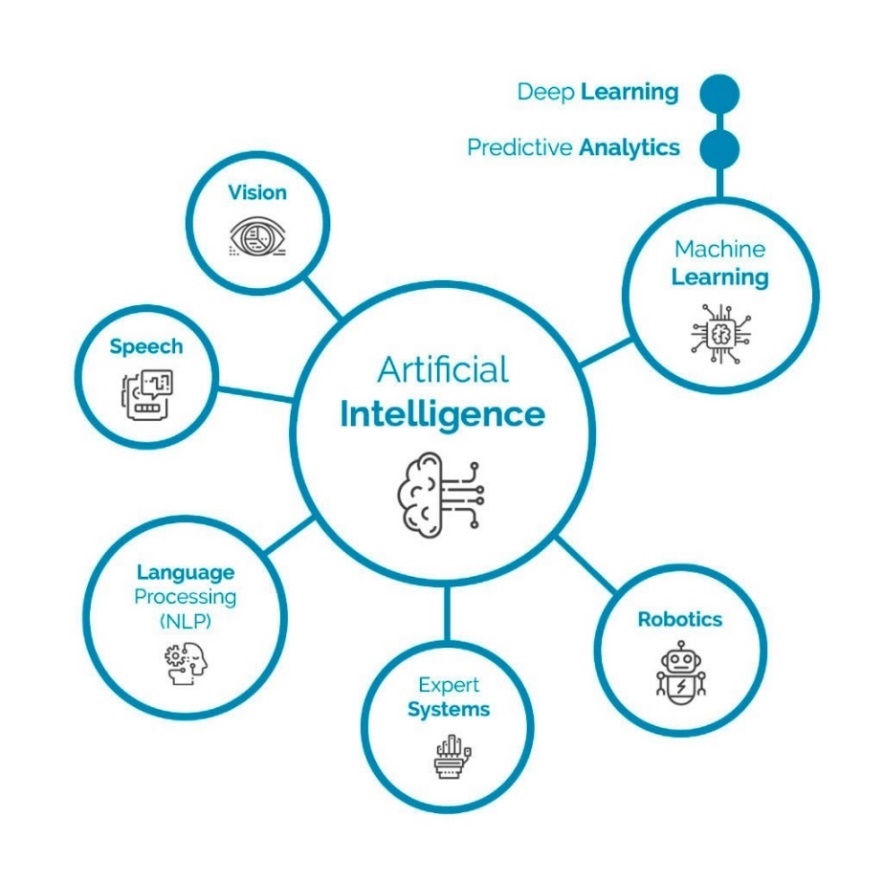
U šestom poglavlju opisan je praktični dio ovog diplomskog rada. Na danom skupu podataka obrađeni su osnovni koraci u strojnom učenju te su primijenjene logistička regresija i neuronske mreže za *multiclass* klasifikaciju uz usporedbu dobivenih rezultata.

# STROJNO UČENJE

Strojno učenje (engl. *Machine learning*, ML) grana je umjetne inteligencije čiji je cilj da računalo ili stroj oponaša ljudsko ponašanje odnosno da donosi zaključke o danom problemu i uči iz iskustva.

## Umjetna inteligencija

Umjetna inteligencija (UI) područje je računalne znanosti koje razvija inteligentne strojeve ili alate koji se ponašaju jako slično kao čovjek. Cilj umjetne inteligencije je razviti sustav koji će se samostalno snalaziti u novim situacijama, koristeći vlastitu „inteligenciju“ što znači bez intervencije korisnika.[1]  
Slika 2.1 prikazuje podjelu umjetne inteligencije na različita područja.



Slika 2.1. Područja umjetne inteligencije [2]

UI temelji se na algoritmima. Oni predstavljaju upute koje stroj može izvoditi. Često se složeniji algoritmi stvaraju tako da se kombinira više jednostavnijih. Mnogi algoritmi UI imaju mogućnost da se samostalno poboljšavaju tako što uče nove strategije, koje su se pokazale kao dobre u prethodnim iskustvima, ili mogu stvarati i neke nove algoritme.

Izazov s kojim se susreće svaka vrsta umjetne inteligencije svakako je ograničenost kognitivnih sposobnosti strojeva i njihove arhitekture. Zbog toga postoje dva tipa modela: strukturalni i funkcionalni. Strukturalnim modelom nastoji se imitirati operacije uma kao što su rasuđivanje i logika. S druge strane, funkcionalni model odnosi se na povezivanje podataka s onim što je izračunato.

## Strojno učenje

Samo strojno učenje odnosi se na programiranje računala da koristi već gotove podatke za poboljšavanje svog rada tako što uči iz njih te prepoznaje obrasce ili uzorke. Računalo koristi algoritme, pomoću kojih uči, što rezultira time da računalo samo, automatski može obraditi nove podatke i dati rezultate obrade preko stvorenih modela.

Naziv strojno učenje prvi je počeo koristiti Arthur Samuel koji je 1952. osmislio mali program za igranje šaha koji strojno učenje opisuje kao „područje koje računalu omogućuje učenje bez da se eksplicitno programira“. Tom Mitchell daje i moderniju definiciju: „Kaže se da računalni program uči iz iskustva E s obzirom na neku klasu zadataka T i mjeru uspješnosti P, ako se njegova uspješnost na zadacima iz T, mjerena mjerom P, poboljšava s iskustvom E“. Zatim se tijekom 60-ih i 70-ih počinje javljati interes za klasifikaciju i prepoznavanje uzoraka kroz razne knjige i radove, a tako se sve više razvija i strojno učenje kakvo poznajemo danas.

## Odabir modela i funkcije za strojno učenje

Proces strojnog učenja temelji se na nekoliko koraka:

1. *Exploratory Data Analysis* (EDA)

* Odluka o tome kakvi podaci se koriste za treniranje računala
* Prikupljanje podataka za treniranje – podaci moraju imati ulazne (po potrebi i pripadne izlazne vrijednosti)
* Analiza podataka (traženje vidljivih poveznica među podacima) i vizualizacija

1. Predobrada (engl. *preprocessing*) i *feature engineering*

* Podaci se upisuju u vektore značajki (engl. *feature vector*), pripremaju se za korištenje tako da se prilagođavaju potrebama problema koji se rješava (mijenja im se format ili se stvaraju nove značajke), set podataka dijeli se na dio za treniranje i dio za testiranje i validacijski dio po potrebi

1. Treniranje modela i validacijske strategije

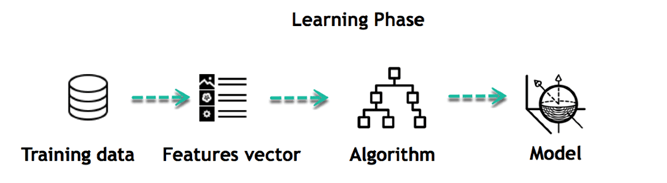
* Odabir algoritama i definiranje funkcija potrebnih za strojno učenje
* Pokretanje algoritma i stvaranje modela (optimizacija parametara potrebnih za preciznost)

1. Metrike i *post-processing* tehnike

* Testiranje dobivene funkcije nad testnim podacima – računa se preciznost dobivenih rezultata
* Ispravljanje mogućih pogrešaka i poboljšanje modela ukoliko je potrebno (u slučaju male preciznosti)

Slika 2.2 prikazuje faze strojnog učenja, odnosno najvažnije postupke za strojno učenje.

Kako bi strojno učenje bilo uspješno provedeno na nekom skupu podataka, potrebno je odabrati dobar algoritam koji uči iz podataka te onda stvoriti dobar model.[3] Model predstavlja ono što je naučeno algoritmom kroz dobro postavljenu hipotezu koja se označava s i ona se opisuje polinomom određenog stupnja.



Slika 2.2. Faze strojnog učenja [4]

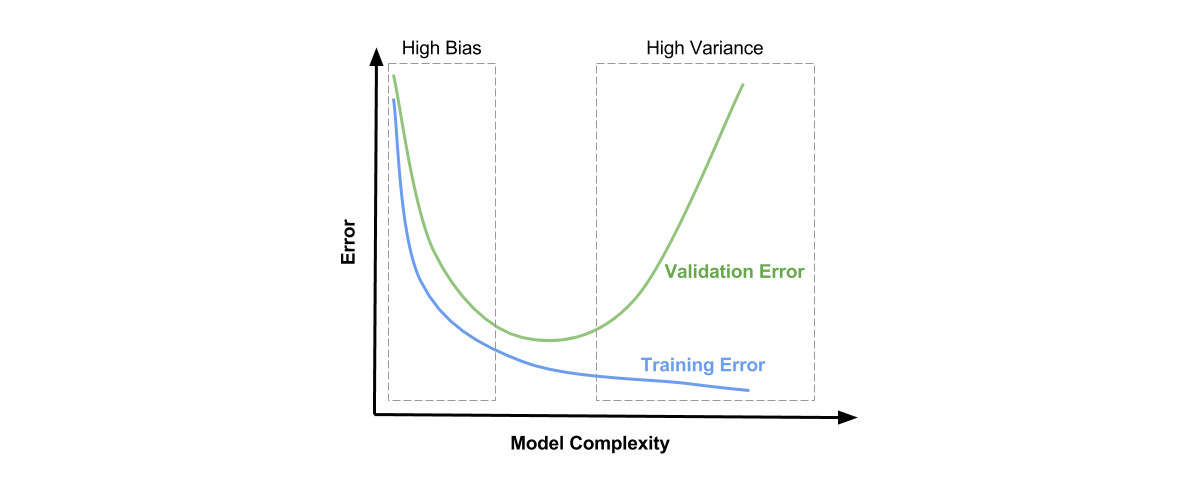
Podaci koji se koriste opisani su vektorom značajki koji je definiran kao , gdje je *X* oznaka za skup ulaznih vrijednosti, a pojedina ulazna vrijednost odnosno značajka. Ti podaci prikupljaju se mjerenjem ili praćenjem onoga što želimo obrađivati algoritmom. Oznakom *m* označava se broj primjera za treniranje, a *n* je dimenzija vektora.

Strojno učenje usko je vezano i uz druga polja kao što su npr. statistika i optimizacija. Kako bi provjerili koliko je dobro postavljena hipoteza, moramo imat funkciju koštanja (engl. *cost function*) koja se označava s , gdje su parametri modela. Funkcija koštanja daje nam razliku između predviđene izlazne vrijednosti i stvarnog izlaza te se još označava i kao pogreška (engl. *error*). Cilj je minimizirati tu funkciju i na taj način optimizirati povezanost modela i sustava koji taj model predstavlja. Strojno učenje tako zapravo teži generalizaciji, odnosno želi se minimizirati gubitak na novim, nepoznatim primjerima, a blisko statistici zaključke donosi na temelju uzoraka te opisuje koliko je dobro postavljen model.

Postupak odabira najboljeg modela nazivamo odabir modela (engl. *model selection*). [5] Budući da je složenost modela određena hiperparametrima modela, o njima i njihovoj optimizaciji ovisi sam odabir. [6] U većini slučajeva, bolja je odluka korištenje jednostavnijih modela koji su lakši za razumijevanje, a i bolje obavljaju već spomenutu generalizaciju. Također, takve je modele lakše naučiti jer imaju manje parametara za optimizaciju, a uz to je lakše iz njih izvesti pravila. Kako bi uspješno odabrali model, podatke je potrebno podijeliti na tri dijela: trening (~60% podataka), *cross validation* (~20% podataka) i testni dio (~20% podataka) te odaberemo stupanj polinoma kojim želimo definirati hipotezu. Za svaki stupanj polinoma računamo pri čemu koristimo *cross validation* dio. Tražimo onaj s minimalnom funkcijom i njega odabiremo kao stupanj koji ćemo koristiti za hipotezu, odnosno model. Problemi koji se mogu javljati su prenaučenost (engl. *overfitting* ili *high variance*) te podnaučenost (engl. *underfitting* ili *high bias*). *Overfitting* se javlja kada model u obzir uzima i detalje i šum u setu podataka za treniranje tako da oni negativno utječu na ponašanje modela za nove podatke i njegovu mogućnost generalizacije, odnosno stupanj polinoma prevelik je za podatke koje imamo. S druge strane, *underfitting* znači da model ne može modelirati ni trening podatke niti generalizirati nove podatke što rezultira lošim performansama. To znači da podatke opisujemo koristeći polinom malog stupnja, a zapravo trebamo veći stupanj da bi ih točno opisali. Ovaj problem je lakše rješiv jer ga je lako detektirati raznim metrikama za izvođenje. [7] Kako bi spriječili da dođe do prenaučenosti ili podnaučenosti, koriste se razne krivulje učenja (engl. *learning curves*) iz kojih se jasno vidi o kojem se problemu radi. Takve krivulje dobijemo kada iscrtamo funkciju koštanja (*error*) za trening set – i *cross validation* set – kao funkciju u ovisnosti o broju primjera *m*.  
Općenito vrijedi:

* Ako se radi o podnaučenosti (*high bias*) – i i imaju velike vrijednosti, također
* Ako se radi o prenaučenosti (*high variance*) – mala vrijednost, ima puno veću vrijednost od

Slika 2.3 prikazuje kako izgledaju krivulje u slučajevima podnaučenosti i prenaučenosti.



Slika 2.3. Krivulje učenja [8]

Ukoliko imamo velike greške za novi set podataka, neka od rješenja su:

* Za ispravljanje *high variance* – dodavanje novih primjera, smanjiti broj značajki
* Za ispravljanje *high bias* – dodavanje dodatnih značajki, dodavanje polinomnih značajki

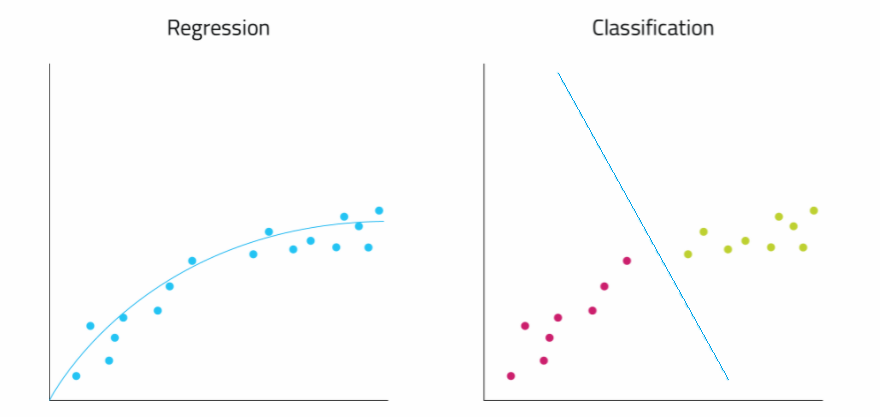
Važno je naglasiti da se pri dizajniranju cijelog sustava mora paziti i na veličinu pogreške. Za strojno učenje postoje različite metrike za analizu rada modela. Uvijek se kreće od nekog osnovnog, malog algoritma koji se brzo implementira. Njega se testira na *cross validation* setu podataka te se provjeravaju krivulje učenja. Prvo se ručno provjeravaju primjeri s potencijalnom greškom, ukoliko je to moguće, a nakon toga kreće se na korištenje metrika. Osnovne metrike su točnost (engl. *accuracy*)*,* preciznost (engl. *precision*) i odziv (engl. *recall*) koje se računaju različito, ovisno o tome o kojoj se vrsti strojnog učenja radi, a i o samim potrebama i zadacima.

Za strojno učenje jako je važno na koji se način prikupljaju podaci i u kakvom su oni obliku. Potrebno je prilagoditi format podataka kako bi postojalo nekakvo općenito pravilo po kojem ih računalo analizira. Tako prilagođeni podaci koriste se zatim u algoritmima koji su definirani pomoću modela za strojno učenje, kriterija za evaluaciju te postupka za optimizaciju svih korištenih parametara. [9] Ovisno o algoritmima i načinu provedbe strojnog učenja, definirane su dvije osnovne vrste strojnog učenja koje su opisane u nastavku.

## Nadzirano učenje

Nadzirano učenje (engl. *supervised learning)* odnosi se na strojno učenje pri kojem računalo uči iz već postojećih primjera koji imaju oznake. To znači da se analiziraju parovi ulaz-izlaz iz skupa podataka, računalo uočava uzorke i poveznice te se definira funkcija koja mu omogućava da za neku novu ulaznu vrijednost daje ispravnu izlaznu vrijednost. [10]  
Hipoteza se definira kao funkcija koja primjerima dodjeljuje oznake (engl. *labels*): . *X* je oznaka za skup ulaznih vrijednosti, a ,kako je prethodno definirano, pojedina ulazna vrijednost, dok je *Y* skup izlaznih vrijednosti gdje se s označava pojedina izlazna vrijednost. Par () naziva se primjer za treniranje (engl. *training example*), a lista od *i=1,…,m* primjera je skup za treniranje (engl. *training set*). Važno je naglasiti da se *i* odnosi na indeks, a ne na eksponent. Hipoteza *h(x)* mora omogućavati da se za dani *x* dobije dobra procjena odgovarajuće vrijednosti *y*. Kažemo da je hipoteza *h* konzistentna s primjerom za učenje (*x, y*)ako i samo ako vrijedi: *h(x)=y*.

Postoje dva osnovna tipa nadziranog učenja, a to su klasifikacija i regresija. Klasifikacijom se određuje pripadnost nekoj klasi, odnosno *y* može poprimiti samo nekoliko diskretnih vrijednosti (ona je detaljno opisana u poglavlju 3), dok se regresijom radi predviđanje neke kontinuirane izlazne vrijednosti. Njihova razlika vizualno je prikazana na slici 2.4



Slika 2.4. Razlika regresije i klasifikacije [11]

## Nenadzirano učenje

Za razliku od nadziranog učenja, kod nenadziranog učenja (engl. *unsupervised learning*) problemu se može pristupiti tako da ne znamo točno kakav rezultat očekujemo nego tražimo nekakvu strukturu u podacima. Skup podataka sastoji se samo od ulaznih vrijednosti *X* te se traži veza među njima bez da postoje izlazne varijable *Y*. Nenadzirano učenje omogućava obavljanje složenijih zadataka u odnosu na nadzirano, ali može biti više nepredvidljivo.

Najpoznatije vrste nenadziranog učenja su PCA (engl. *Principal Component Analysis*) te analiza grozdova ili grupiranje (engl. *clustering*). *Clustering* se koristi na način da se podaci dijele u grupe, odnosnoklastere na način da se zajedno grupiraju oni podaci koji imaju zajedničke atribute te između kojih postoji nekakva poveznica. [12] *Clustering* sam po sebi nije jedan algoritam, ali svaki algoritam koji se koristi za taj postupak omogućava pronalaženje strukture u podatcima. Najčešće korišten *clustering* algoritam je *k-means* algoritam, gdje *k* označava broj korištenih klastera.

Ovakav pristup također omogućava i detekciju anomalija (engl. *anomaly detection*), odnosno podataka koji ne pripadaju nijednoj grupi. Ponekad je teško odabrati treba li se koristiti detekcija anomalija ili nadzirano učenje pa postoji nekoliko smjernica po kojima se dobro voditi pri tom odabiru:

|  |  |
| --- | --- |
| **Detekcija anomalija** | **Nadzirano učenje** |
| * Mali broj pozitivnih primjera (y=1 označava anomaliju) * S obzirom na mali broj pozitivnih primjera teško je naučiti iz njih jer buduća anomalija neće ličiti na prethodnu pa je lakše modelirati negativne primjere * Veliki broj negativnih primjera (y=0) koje onda možemo koristiti za stvaranje modela | * Velik broj i pozitivnih i negativnih primjera * Koristi puno pozitivnih i negativnih primjera za treniranje jer postoji dovoljan broj pozitivnih za razliku od detekcije anomalija |
| **Primjeri korištenja:** detekcija prevara, greške u proizvodnji, nadzor strojeva | klasifikacija emailova (*spam* poruka ili ne), predviđanje vremena |

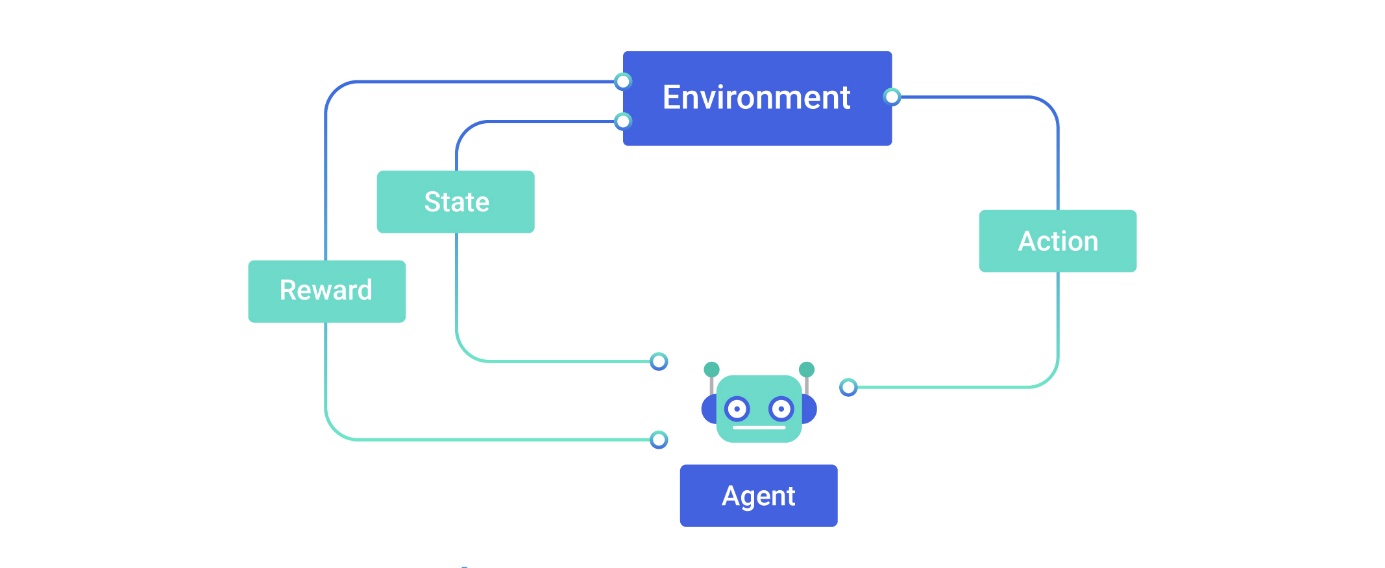
## Polu-nadzirano učenje

Polu-nadzirano učenje (engl. *semi-supervised learning*) vrsta je strojnog učenja koje za treniranje kombinira malu količinu označenih podataka i veliku količinu neoznačenih podataka. [13] Koristi se za setove podataka nad kojima se radi klasifikacija što znači da trebamo algoritme nadziranog učenja, ali želimo trenirati model bez da označavamo sve one primjere koji su bez oznake za što nam onda trebaju algoritmi nenadziranog učenja. Jedan od načina koji se često koristi je korištenje klasterai klasifikacije. Pomoću *clusteringa* pronalaze se najvažniji podaci te ih onda možemo označiti i koristiti u klasifikaciji. Na ovaj se način može postići poboljšanje što se tiče točnosti modela.

## Podržano učenje

Podržano učenje je nešto drugačiji tip učenja od prethodno spomenutih. Odnosi se na korištenje inteligentnih agenata koje se trenira da poduzimaju akcije, u svom okruženju, s ciljem da se dođe do onoga što korisnik želi. Koristi u teorijama igara, robotici, statistici, navigaciji itd.

Slika 2.5 prikazuje kako se odvija podržano učenje, a da bi ono bilo uspješno provedeno potrebne su tri glavne komponente: agent (donosi odluke), okruženje (sve što je u interakciji s agentom) i akcije (radnje koje agent radi i o kojima donosi odluke). Samo učenje odvija se kroz sekvence odluka koje donosi model, na način da koristi pokušaje i pogreške da bi došao do rješenja tako što se nagrađuju ili kažnjavaju odluke koje su donesene. [14] Svaki agent nastoji maksimizirati ukupne nagrade tijekom određenog vremena. Cilj će dostići mnogo efikasnije ako pronađe dovoljno dobru strategiju pri donošenju odluka, a pri tome ne dobiva nikakve sugestije ili savjete vezano za odluke.



Slika 2.5. Podržano učenje [15]

Ono što je izazov kod ovakve vrste strojnog učenja prvenstveno je postavljanje okruženja za simulacije jer ono jako ovisi o zadatku na kojem se radi. Također, skaliranje i popravljanje modela još je jedan izazov jer za komunikaciju ne postoji ništa osim nagrade i kazne. Uz to, agent može odraditi zadatak, ali ne na optimalan način ili da optimizira nagradu, a ne odradi onaj zadatak za koji je dizajniran. Svakako, podržano učenje jako je korisno i sve se više koristi u raznim područjima, ali je potreban trud da bi se sve uspješno odradilo.

# KLASIFIKACIJA

Klasifikacija je vrsta nadziranog učenja kod koje se podacima odnosno primjerima pridjeljuje klasa kojoj oni pripadaju. Klase su zapravo već spomenute oznake koje pripadaju skupu *Y*, a s obzirom da se radi o nadziranom učenju oznake klasa unaprijed su poznate. One se povezuju s pripadnim podacima preko funkcija za mapiranje, ali u ovom slučaju te izlazne vrijednosti poprimaju diskretne vrijednosti. Cilj klasifikacije razviti je dovoljno dobar model koji će točno odrediti kojoj klasi pripadaju novi podaci koji nisu bili korišteni pri treniranju algoritma.

Postoje dvije vrste klasifikacija koje su opisane u nastavku.

## Binarna klasifikacija

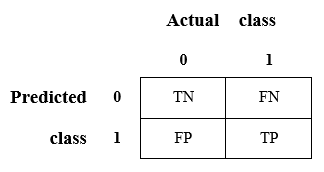
Ako se klasifikacija ograniči na samo dvije klase, govorimo o binarnoj klasifikaciji. Tada vrijedi: , gdje y=1 označava da je primjer pozitivan za jednu od dvije klase (pripada toj klasi), dok y=0 označava da je primjer negativan za tu klasu, odnosno primjer ne pripada toj klasi. [16] Takav se klasifikator nazivan binarni klasifikator. Hipoteza za binarnu klasifikaciju ima oblik: i ona nam daje vjerojatnost da je izlazna vrijednost 1. Na primjer, uvjeti mogu biti postavljeni na slijedeći način:

Vrijednost 0.5 naziva se vrijednosni prag (engl. *threshold*), a mjesto gdje je naziva se granica odluke (engl. *decision boundary*), koja ne mora biti samo linearna nego može poprimiti i druge oblike.

Evaluacija binarne klasifikacije provodi se korištenjem već spomenutih metrika: točnost, preciznost i odziv. Kako bi se metrike koristile, prvenstveno je potrebno definirati matricu zabune (engl. *confusion matrix*). Ona se sastoji od 4 kategorije u koje se dijele dobiveni rezultati:

* TP (engl. *true positive*) – primjeri koji su točno klasificirani kao pozitivni
* FP (engl. *false positive*) – primjeri koji su klasificirani kao pozitivni, a nisu pozitivni
* FN (engl. *false negative*) – primjeri koji su klasificirani kao negativni, a nisu negativni
* TN (engl. *true negative*) – primjeri koji su točno klasificirani kao negativni

*Confusion matrix* prikazana je na slici 3.1.



Slika 3.1. Confusion matrix

Ukupan broj primjera označen je s *N*,a na osnovu matrice zabune, metrike se definiraju na sljedeći način. [16]

### Točnost

Točnost je omjer ispravno klasificiranih primjera i ukupnog broja primjera. Definira se izrazom (3.1)

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

Problem s točnošću javlja se kada imamo disbalans u podacima. Takva pojava naziva se *skew* u podacima i znači da je distribucija podataka nagnuta na lijevu ili desnu stranu, odnosno nije simetrična. Tada se može dogoditi da imamo jako visok udio primjera jedne klase (npr. pozitivne) pa će klasifikator, koji svaki novi primjer klasificira kao pozitivan, imati visoku točnost, a zapravo se radi o jako lošem modelu. [17]

### *Preciznost*

Preciznost je omjer ispravno klasificiranih pozitivnih primjera i ukupnog broja primjera koji su klasificirani kao pozitivni te se definira izrazom (3.2)

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |

Odgovara na pitanje koliko je primjera zapravo točno klasificirano od svih koje je model klasificirao kao pozitivne.

### Odziv

Odziv je omjer ispravno klasificiranih pozitivnih primjera i ukupnog broja primjera koji su zapravo pozitivni. Definiran je izrazom (3.3)

|  |  |  |
| --- | --- | --- |
|  |  | (3.3) |

Odgovara na pitanje koliko je primjera model točno klasificirao za pozitivnu klasu. Ako, recimo, model stalno predviđa *y*=0, odziv će biti jednak nuli.

U praksi je teško postići i visoku preciznost i visok odziv pa se radi tzv. *trade off*. To znači da ako, primjerice, želimo izbjeći FN rezultate, moramo imati veći odziv, a manju preciznost.

Posebna metrika koja je izvedena iz prethodno navedenih osnovnih metrika je F1 *score*. On je težinski prosjek preciznosti i odziva. F1 u obzir uzima i FP i FN i često je dosta korisniji od točnosti, posebno kod nejednake distribucije klasa. Definira se izrazom (3.4)

|  |  |  |
| --- | --- | --- |
|  |  | (3.4) |

Metode koje se najčešće koriste za binarnu klasifikaciju su:

* Stablo odlučivanja
* Neuralne mreže
* Bayesian klasifikacija
* Support Vector Machines

## *Multiclass* klasifikacija

Često se javlja potreba da se primjeri klasificiraju u više od dvije klase i to nazivamo višeklasnom klasifikacijom (engl. *multiclass classification*). Tada se pripadna oznaka klase za neki primjer prikazuje kao *K* – dimenzijski vektor, . Općenito, postoji K klasa koje se označavaju s te vrijedi definicija (3.5):

|  |  |  |
| --- | --- | --- |
|  |  | (3.5) |

Npr. znači da primjer pripada klasi .

Metode koje se najčešće koriste za *multiclass* klasifikaciju su:

* Naive Bayes
* KNN (*K-nearest neighbours*)
* Neuralne mreže
* Logistička regresija (s jednom od metoda opisanih u nastavku)

Najčešće se problemi *multiclass* klasifikacije rješavaju tako da se problem svodi na binarnu klasifikaciju, a za to se koriste *one-vs-rest* (još se naziva i *one-vs-all*) i *one-vs-one* metode. [18]

### One-vs-rest

Kod ove metode odaberemo jednu klasu koju ćemo trenirati, dok sve ostale klase okupimo u drugu klasu. Tako smo dobili problem binarne klasifikacije i postupak ponavljamo za sve postojeće klase, odnosno imamo onoliko binarnih klasifikacija koliko imamo klasa.

Primjer: imamo 3 klase (red, blue, green) što znači da imamo 3 binarne klasifikacije:

* Binarna klasifikacija 1: red – [blue, green]
* Binarna klasifikacija 2: blue – [red, green]
* Binarna klasifikacija 3: green – [red, blue]

Hipoteza definirana izrazom (3.6) kod ovakve se metode postavlja za svaki binarni klasifikator.

|  |  |  |
| --- | --- | --- |
|  | *=,* *i=1,2,3,…,k* | (3.6) |

gdje *k* označava broj klasa.

Klasifikator se trenira za svaku klasu *i* da predviđa vjerojatnost da je *y*=*i*. Za svaki novi ulaz *x* odabire se klasa *i* koja maksimizira kako bi se predvidjelo kojoj klasi pripada taj ulaz.

Ova je strategija popularna, ali ima određene probleme. Glavni problem je da klase mogu biti jednoliko raspoređene u trening setu, ali s obzirom da treniramo jednu po jednu klasu, set negativnih će biti puno veći od pozitivnih što može loše utjecati na model i samu klasifikaciju. Isto tako proces može dugo trajati za velike setove podataka, spore modele ili veliki broj klasa.

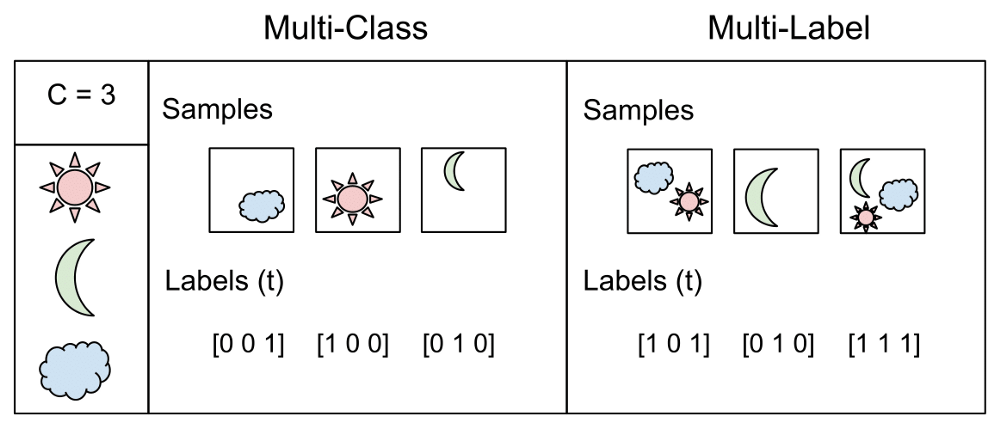
### One-vs-one

Kod ove metode imamo binarnih klasifikacija, gdje je *K* broj klasa. To znači da svaki par klasa treniramo jednom.

Primjer: imamo 4 klase (red, blue, green, yellow) što znači da imamo 6 binarnih klasifikacija:

* Binarna klasifikacija 1: red – blue
* Binarna klasifikacija 2: red – green
* Binarna klasifikacija 3: red – yellow
* Binarna klasifikacija 4: blue – green
* Binarna klasifikacija 5: blue – yellow
* Binarna klasifikacija 6: green – yellow

Često dolazi do zabune kada se govori o *multi-class* i o *multi-label* klasifikaciji. Ove dvije vrste se jako razlikuju i ne smije ih se miješati. *Multi-label* odnosi se na primjere koji mogu pripadati u više klasa, dok kod *multi-class* svaki primjer pripada točno određenoj klasi, što je prikazano na slici 3.2.



Slika 3.2. Razlika multiclass i multilabel klasifikacija [19]

Postoji nekoliko metoda za provođenje *multilabel* klasifikacije:

* Transformacija u problem binarne klasifikacije – korištenje *binary relevance*metode (treniranje jednog binarnog klasifikatora za svaku klasu)
* Transformacija u problem *multiclass* klasifikacije – korištenje *label powerset* transformacije (koristi se jedan binarni klasifikator za svaku kombinaciju klasa koje postoje)
* Korištenje više *multiclass* klasifikatora za stvaranje *multilabel* klasifikatora

S obzirom na paradigme učenja, tehnike koje se koriste za *multilabel* klasifikaciju mogu se podijeliti na *batch* ili *online* ML. Kod *batch* algoritama, svi podaci moraju biti dostupni prije samog korištenja te se koristi cijeli trening set. S druge strane, *online* algoritmi inkrementalno sastavljaju svoje modele u sekvencijalnim iteracijama.

# LOGISTIČKA REGRESIJA

Logistička regresija klasifikacijski je algoritam, ali se naziva regresija jer se koriste jako slične tehnike kao kod linearne regresije. Jedan je od najšire korištenih algoritama u strojnom učenju. Pri takvoj klasifikaciji, izlazne su vrijednosti diskretne (*y*=0 ili *y*=1), a vrijednost hipoteze . Logistička regresija ima svoju funkciju koštanja te se može koristiti i za *multiclass* klasifikaciju što će biti opisano u nastavku.

Kako bi definirali hipotezu, prvo je potrebno definirati aktivacijsku funkciju koja se naziva *sigmoid* (*logistic*) funkcija. Njezina definicija dana je izrazom (4.1), a graf funkcije prikazan je na slici 4.1.

|  |  |  |
| --- | --- | --- |
|  | , , | (4.1) |

gdje je *z* neki realan broj.



Slika 4.1. Graf sigmoid funkcije [20]

Kao što je prikazano, *sigmoid* se približava 0 kada *z* ide u -∞ te se približava 1 kada *z* ide u +∞, odnosno *g(z)* poprima vrijednosti između 0 i 1. Hipoteza se definira izrazom (4.2)

|  |  |  |
| --- | --- | --- |
|  |  | (4.2) |

Na taj se način ostvaruje traženi uvjet da je . Kao što je već spomenuto, dobivena vrijednost odnosi se na vjerojatnost da je *y*=1, što se označava kao:

|  |  |  |
| --- | --- | --- |
|  |  | (4.3) |

Definicija (4.3) interpretira se kao „vjerojatnost da je *y*=1 za dane značajke *x* s parametrima “

Funkcija koštanja za pojedini primjer definirana je izrazom (4.4).

|  |  |  |
| --- | --- | --- |
|  |  | (4.4) |

Ovakva definicija funkcije ima nekoliko dobrih karakteristika. Iz definicije zaključujemo da je *Cost*=0 ako je , odnosno, ako hipoteza točno predviđa izlaznu vrijednost, nemamo nikakav gubitak. S druge strane, kada npr. za y=1 , tada . Takav je zaključak intuitivan: ako je , odnosno , a zapravo vrijedi *y*=1, algoritam će biti penaliziran funkcijom koštanja velike vrijednosti. Definicija se može zapisati i na jednostavniji način (4.5) zbog toga što imamo samo dvije vrijednosti koje *y* poprima.

|  |  |  |
| --- | --- | --- |
|  |  | (4.5) |

Ukupna funkcija koštanja za cijeli trening set tada se može definirati izrazom (4.6).

|  |  |
| --- | --- |
|  | (4.6) |

Ovako definirana funkcija koštanja je konveksna i koristi se uvijek za postavljanje modela logističke regresije.

Kod logističke regresije može se pojaviti i problem prenaučenosti pa se uz već spomenuta rješenja koristi i regularizacija. Ideja iza same regularizacije korištenje je manjih parametara te se tako stvara jednostavnija hipoteza i manja je vjerojatnost da će doći do prenaučenosti. Regularizacija se provodi dodavanjem regularizacijskog izraza s parametrom α na funkciju koštanja (4.7).

|  |  |
| --- | --- |
|  | (4.7) |

Parametar α naziva se regularizacijski parametar i on kontrolira odnos između dva cilja: uklapanje podataka u hipotezu i zadržavanje parametara u malim vrijednostima. α ne smije biti prevelike vrijednosti jer algoritam može postati podnaučen i neće dobro raditi čak ni za trening set. Ovaj izraz ne penalizira , ali to ne predstavlja značajnu razliku.

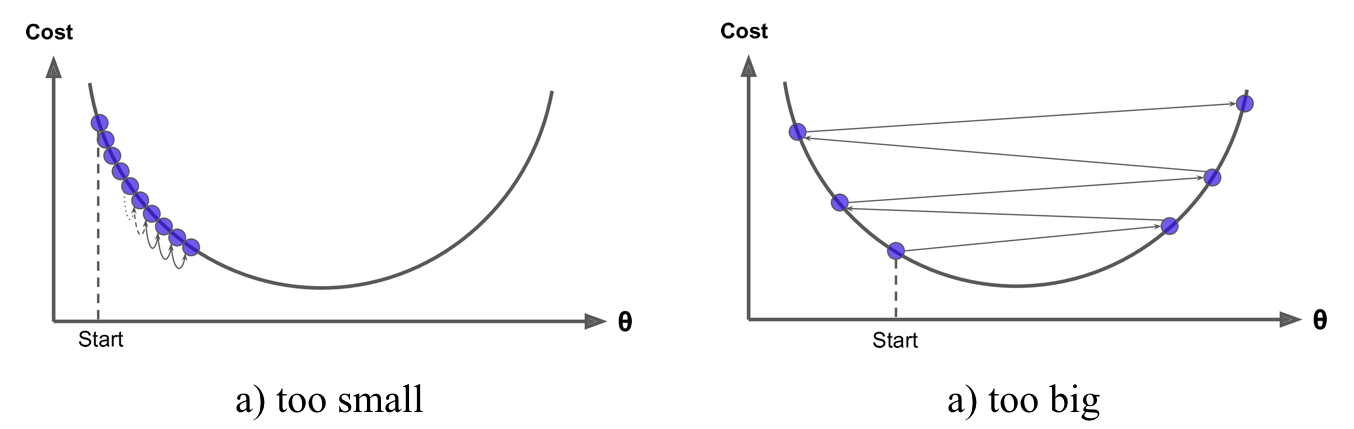
## Gradijentni spust

Osim hipoteze i funkcije koštanja, potrebno je odabrati parametre , nakon čega je moguće raditi predviđanja korištenjem dane hipoteze. Kako bi parametri bili dobro postavljeni, potrebno je pronaći takve parametre za koje je vrijednost minimalna. Tada za neki novi primjer *x* možemo raditi predviđanja kako bi dobili izlaznu vrijednost koristeći hipotezu (imamo mjere koje govore koliko dobro ona zadovoljava) te dobivene parametre. Za minimizaciju funkcije koštanja koristi se gradijentni spust (engl. *gradient decent*). Općenito, gradijentni spust odvija se na način da se uzmu neki početni parametri te se računa . Postupak se ponavlja tako što se parametri mijenjaju kako bi se došlo do minimalne vrijednosti što je prikazano na slici 4.2.



Slika 4.2. Gradijentni spust [21]

Za gradijentni spust posebno je važna stopa učenja (engl. *learning rate*) koja se označava s . Ona kontrolira koliki je korak pri gradijentnom spustu. Pri izboru stope učenja treba voditi računa o eventualnim problemima koji mogu nastati. Ako koeficijent premali, gradijentni spust bit će spor i previše će koraka trebati do minimuma. S druge strane, ako je prevelik, gradijentni spust može jednostavno preskočiti minimum te se može dogoditi da ne konvergira ili da čak divergira (Slika 4.3). Smjer u kojem idemo prema minimumu određen je parcijalnom derivacijom .



Slika 4.3. Usporedba premale i prevelike stope učenja [22]

Vrijednost može biti fiksna jer gradijentni spust, kada se približava minimumu, automatski radi manje korake (jer se derivacija približava 0) što znači da ne treba mijenjati kroz vrijeme.

Općenita definicija gradijentnog spusta:

Ponavljaj {

}

Parcijalna derivacija za jedan primjer računa se kao (4.8).

|  |  |  |
| --- | --- | --- |
|  |  | (4.8) |

Za logističku regresiju, gradijentni spust zato se definira na sljedeći način:

Ponavljaj {

(simultano ažuriraj sve vrijednosti )

}

mora se smanjivati pri svakoj iteraciji.

Za gradijentni spust također postoji regularizacijski izraz. Kako je već spomenuto, u regularizaciju s ne ulazi pa s njim radimo na poseban način:

Ponavljaj {

}

To znači da se parcijalna derivacija sada definira izrazom (4.9).

|  |  |  |
| --- | --- | --- |
|  |  | (4.9) |

## *Multiclass* logistička regresija

*Multiclass* logistička regresija naziva se još i *multinomial* *logistic regression* i *softmax regression* (SMR). Ona je zapravo generalizacija logističke regresije koja sažima *k* dimenzionalni vektor proizvoljnih vrijednosti u *k* dimenzionalni vektor vrijednosti u rasponu (0, 1). Možemo je koristiti za *multiclass* klasifikaciju, uz uvjet da neki primjer može pripadati samo jednoj klasi. Nešto više o korištenju ovakve vrste logističke regresije u poglavlju o praktičnoj primjeni.

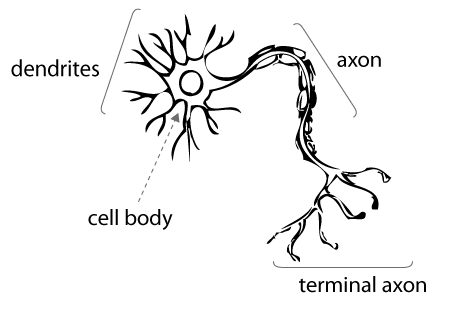
# NEURONSKE MREŽE

Umjetna neuronska mreža (engl. *artificial neural network*, ANN) skup je algoritama čiji je cilj prepoznavanje osnovnih veza među podacima kroz proces imitacije operacija ljudskog mozga, odnosno kroz simulaciju rada neurona u mozgu.

## Osnovni pojmovi

Neuroni su osnovne jedinice živčanog sustava i najsloženije su u ljudskom organizmu. [16] Sastavni dijelovi neurona su (Slika 5.1):

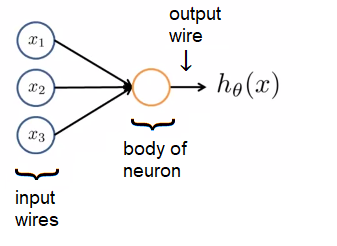
* Dendriti – kraći produžeci koji služe kao „ulazna žica“ (engl. *input wire*) na način da s osjetilnih organa dovode pobudu na tijelo stanice
* Tijelo stanice (engl. *cell body*) – u njemu se nalazi jezgra i kromosomi
* Aksoni – duži produžetak koji služi kao „izlazna žica“ (engl. *output wire*), odnosno prenosi živčane impulse s tijela stanice na druge živčane stanice ili izvršne organe



Slika 5.1. Građa neurona [23]

Kod umjetne neuronske mreže, koristi se zapravo jednostavni model onoga što neuron radi, odnosno neuron se modelira kao logistička jedinica (Slika 5.2). Ponovo imamo *sigmoid* aktivacijsku funkciju (5.1) koja se označava i s *g*(*x*).

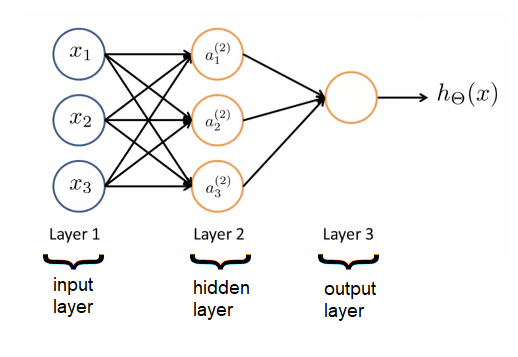
|  |  |  |
| --- | --- | --- |
|  |  | (5.1) |



Slika 5.2. Model neurona

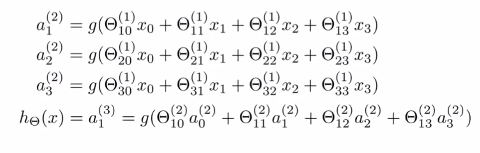
Uz *x1, x2* i *x3* ponekad se dodaje i *x0* koji se naziva *bias unit* za koji uvijek vrijedi *x*0=1.

Umjetne neuronske mreže su slojevite i sastoje se od tri glavna sloja: ulazni sloj (engl. *input layer*), skriveni sloj (engl. *hidden layer*) i izlazni sloj (engl. *output layer*) kao što je prikazano na slici 5.3. Ulazni sloj sadrži značajke *x* iz skupa podataka, izlazni sloj daje finalnu vrijednost *y* koju računa hipoteza , a skriveni sloj sadržava vrijednosti koje ne vidimo u trening setu pa stoga i naziv skriveni.



Slika 5.3. Model neuronske mreže sa slojevima

Za ovakav model, aktivacije i hipoteza prikazane su na slici 5.4.



Slika 5.4 Aktivacija neurona i hipoteza za dani model

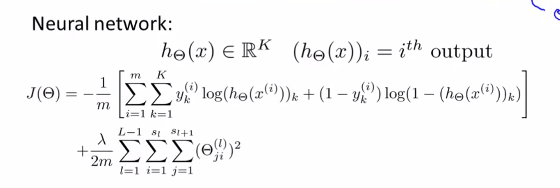
– aktivacija jedinice (neurona) *i* u sloju *j* (aktivacija se odnosi na vrijednost koja je izračunata i dana kao izlaz)

– matrica težina (težinskih faktora ili parametara) koja kontrolira mapiranje iz sloja *j* u sloj *j*+1; ako mreža ima jedinica u sloju *j*, a jedinica u sloju *j*+1, onda će imati dimenzije .

Ako za primjer uzmemo neuronsku mrežu koja rješava problem klasifikacije, podaci za treniranje su u obliku: . *L* je ukupan broj slojeva u neuronskoj mreži, a broj jedinica (bez *bias* jedinice) u sloju *l.* Ako se radi o binarnoj klasifikaciji na kraju se dobije samo 1 izlazna jedinica, dok za *multiclass* klasifikaciju (*K* klasa) imamo *K* izlaznih jedinica.

Neuronske mreže mogu imati različite dijagrame i to se naziva arhitektura neuronskih mreža. Pojam arhitektura odnosi se na način na koji su povezani različiti neuroni.

Za neuronske mreže funkcija koštanja zapravo je generalizacija funkcije koštanja za logističku regresiju. Tako umjesto jedne izlazne jedinice, imamo njih *K* (s tim da *K* može biti i 1 kao što je prethodno spomenuto). Razlika u funkciji koštanja je u sumaciji po izlaznim jedinicama, odnosno imamo funkciju koštanja logističke regresije za svaku izlaznu jedinicu i njih sumiramo (Slika 5.5).



Slika 5.5 Funkcija koštanja za neuronske mreže

## *Backpropagation* algoritam

Uz prethodno definirani način koji se odnosi na *forward propagation* algoritam, drugi algoritam za minimizaciju funkcije koštanja naziva se *backpropagation* (BP)algoritam. Odnosi se na to kako se računa gradijent i cilj mu je, ponovo, pronaći parametar za kojeg je minimalan. Za računanje derivacija koristi se *backpropagation* algoritam. Prvenstveno, definiramo pojmove važne za taj postupak:

* – aktivacija jedinice (čvora) *j* u sloju *l*
* za svaki čvor definiramo izraz – „greška“ čvora *j* u sloju *l*, odnosno greška u aktivaciji čvora

Za svaku izlaznu jedinicu definiramo izraz (5.2).

|  |  |  |
| --- | --- | --- |
|  | , | (5.2) |

Iz ove definicije jasno je da je razlika između onoga što izračuna hipoteza i stvarne vrijednosti u primjeru.

Da bi prikazali rad *backpropagation* algoritma, uzimamo trening set oblika: .

1. Postavi , za sve *i*, *j*, *l*

koristimo za računanje parcijalne derivacije

1. For *i=1* to *m* {

Postavi

Radi *forward propagation* kako bi izračunali za *l=*2, 3,…, *L*

Koristeći , izračunaj

Izračunaj

}

1. Nakon što se završi *for* petlja, računa se sljedeće:

Kada je *j*=0 to odgovara *bias* jedinici pa nema ni regularizacijskog izraza.

Na kraju možemo zaključiti da vrijedi (5.3).

|  |  |  |
| --- | --- | --- |
|  |  | (5.3) |

# *MULTICLASS* KLASIFIKACIJA CYBER-NAPADA

U ovom poglavlju opisana je praktična primjena svih teorijskih područja koja su opisana u prethodnim poglavljima. Korišteni set podataka preuzet je sa stranice Kaggle koja omogućava korisnicima preuzimanje, ali i objavljivanje različitih skupova podataka. [24] Uz to, omogućava i različite alate za stvaranje i korištenje modela strojnog učenja, rad s drugim korisnicima te mogućnost sudjelovanja na raznim natjecanjima.

Za cijeli postupak strojnog učenja korišten je Visual Studio Code te Python 3.8.8 gdje je kreirano virtualno okruženje.

Korišteni set podataka zapravo je poboljšana verzija poznatog KDD-99 seta koji je bio namijenjen za natjecanje *The Third International Knowledge Discovery and Data Mining Tools Competition* i taj novi set naziva se NSL-KDD set podataka. [25] On rješava neke od glavnih problema KDD-99 seta, kao što su:

* Ne uključuje redundantne zapise u trening setu (zbog čega klasifikator neće biti pristran prema češćim zapisima)
* Nema dupliciranih zapisa u test setu
* Broj zapisa i u trening i u test setu je dovoljan da se mogu provoditi istraživanja koja su konzistentna i mogu poslužiti za analize i usporedbe, a nije prevelik da bi se trebalo nasumično odabirati manje dijelove

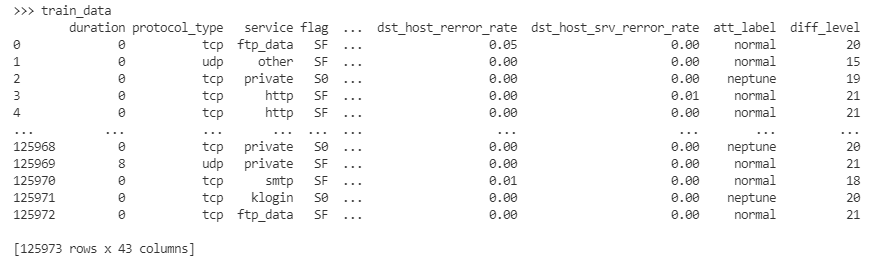
## Korištene biblioteke

Za potrebe ovog rada korištene su biblioteke: numpy, pandas, sklearn, keras, matplotlib i seaborn. Numpy je biblioteka za Python koja omogućava rad s višedimenzionalnim objektima, rad s matematičkim i logičkim manipulacijama, sortiranje, Fourierove transformacije, linearnu algebru, statističke operacije i simulacije i sl. [26] Pandas je softverska biblioteka za Python koja se koristi za analizu i rad s podacima. [27] Za samo strojno učenje korištena je biblioteka sklearn koja nudi različite algoritme za provedbu regresije, klasifikacije te *clusteringa*. Matplotlib i seaborn biblioteke koriste se za iscrtavanje različitih grafova pri analizi podataka.

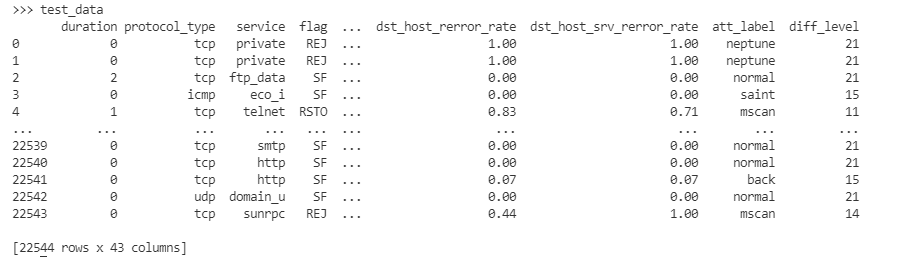
## Analiza podataka

Prvi je korak, u svakom problemu strojnog učenja, eksplorativna analiza podataka (engl. *Exploratory Data Analysis*, EDA). Primarni cilj takve analize podataka je prikazivanje glavnih karakteristika podataka kroz statistiku i grafički prikaz. EDA omogućava da se kroz vizualizaciju uoče poveznice među podacima tako da se mogu stvoriti neke nove značajke ili da se neke značajke izbace jer nemaju nikakvih poveznica s drugima pa samim time nemaju ni preveliku važnost. EDA uvijek kreće od početnog seta podataka koji je na raspolaganju, a odnosi se na oblik podataka i broj značajki. [28]

Set podataka zadrži zapise internetskog prometa koje je pohranila jednostavna mreža za otkrivanje napada. Svaki primjer predstavlja jednu mrežnu vezu sa svojim značajkama. Svaka mrežna veza ima 43 značajke pri čemu se 41 odnosi na sam promet i njegove karakteristike, a posljednje dvije su oznaka napada (*att\_label* – radi li se o nekom napadu ili normalnoj vezi) te ocjena (*diff\_level* – razina značaja same veze). [29] Na priloženim slikama (Slika 6.1 i Slika 6.2) vidljivo je da je broj značajki 43 za svaki primjer i u trening i u testnom setu.



Slika 6.1. Podaci iz trening seta



Slika 6.2. Podaci iz testnog seta

Koristeći pandas biblioteku, potrebno je pregledati sažetak informacija o podacima (prikazani su samo primjeri za svaki tip podatka):

>>>train\_data.info()

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 duration 125973 non-null int64

1 protocol\_type 125973 non-null object

24 serror\_rate 125973 non-null float64

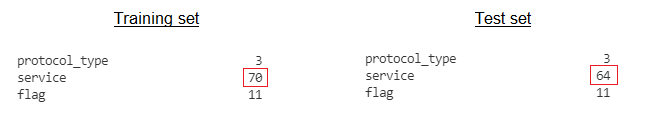
dtypes: float64(15), int64(24), object(4)

Uz to, iz vrijednosti je vidljivo da postoje različite grupe među značajkama koje mogu pomoći pri stvaranju modela, a to su:

* Kategoričke značajke – značajke koje imaju imenovane vrijednosti, odnosno nisu numeričke, npr. *protocol\_type* govori kakav je protokol korišten pri toj vezi (takve značajke potrebno je enkodirati)
* Kontinuirane značajke – npr. *duration*, *src\_bytes*, *dst\_bytes* su *integer* vrijednosti koje mjere ono što se prati i mogu imati različite raspone, odnosno različite vrijednosti (takve značajke potrebno je normalizirati)
* Diskretne značajke – većina ima oznaku *­\_rate* i *float* su vrijednosti s rasponom od 0 do 1, a tu možemo uključiti i binarne značajke s vrijednostima točno 0 ili 1

Nakon toga potrebno je provjeriti ima li *null* i *NaN* vrijednosti. Taj korak je važan jer takve vrijednosti mogu unijeti disbalans u podacima i utjecati na konačan rezultat učenja. U danim podacima nema takvih vrijednosti.

Važan korak je pregled usklađenosti podataka među setovima pomoću train\_data.nunique() i test\_data.nunique(). Ove dvije naredbe omogućavaju ispis broja jedinstvenih vrijednosti za svaku značajku, bez obzira na tip. Pri ispisu, vidljivo je i da *num\_outbound\_cmds* ima samo jednu vrijednost (ta vrijednost je 0) u oba seta pa se može odbaciti jer nema utjecaja i ne pridonosi ništa postupku klasifikacije. Pozornost treba obratiti na kategoričke značajke te značajke koje su binarne, odnosno mogu postići samo vrijednost 0 ili 1. Na slici 6.3 je prikazan rezultat za kategoričke značajke



Slika 6.3. Jedinstvene vrijednosti za kategoričke značajke

Ovakav rezultat pokazuje da u trening setu imamo 6 tipova usluga manje i zbog toga je potrebno te vrijednosti dodati kao prazne stupce kako bi se ispravno izvršilo enkodiranje tijekom pripreme podataka za stvaranje modela. Što se tiče binarnih značajki, ispisom je vidljivo da značajka *su\_attempted* ima vrijednosti 0, 1, ali i 2 pa je to potrebno ispraviti na neki način. Sve ovo obavlja se u postupku *preprocessinga* koji slijedi nakon analize.

S obzirom da se radi o *multiclass* klasifikaciji, potrebno je definirati klase na koje se dijele dani podaci odnosno mrežne veze. Imamo 5 glavnih klasa napada (4 definirane u nastavku i normalno povezivanje):

* Napad uskraćivanjem resursa (engl. *Denial of Service*, DoS) – napad kod kojeg se pokušava zaustaviti mrežni promet prema ciljnom sustavu i od tog sustava, uspostavlja se abnormalna količina prometa, koji sustav ne može podnijeti, pa se sustav isključuje da bi se zaštitio i zbog toga mu se ne može pristupiti
* *Probe* (*surveillance*) napad – napad kod kojeg se pokušava dobiti informacije s mreže, cilj je ukrasti važne podatke, bilo da se radi o osobnim podacima klijenata ili bankarskim podacima
* *User to Root* (U2R) napad – napad koji započinje s normalnim korisničkim računom te se preko njega pokušava pristupiti sustavu ili mreži kao *root* korisnik, napadač pokušava iskoristiti ranjivosti u sustavu kako bi stekao *root* privilegije i pristup važnim podacima
* *Remote to Local* (R2L) napad – napad kod kojeg se pokušava ostvariti lokalni pristup udaljenom stroju, napadač nema lokalni pristup sustavu/mreži i pokušava preko napada doći do sustava, odnosno mreže

Definiramo koji napadi pripadaju kojoj klasi:

dos\_attacks = ['apache2', 'back', 'land', 'neptune', 'mailbomb', 'pod',

               'processtable', 'smurf', 'teardrop', 'udpstorm']

probe\_attacks = ['ipsweep', 'mscan', 'nmap','portsweep', 'saint', 'satan']

u2r\_attacks = ['buffer\_overflow', 'loadmodule','perl', 'ps', 'rootkit', 'sqlat tack', 'xterm', 'httptunnel']

r2l\_attacks = ['ftp\_write', 'guess\_passwd', 'imap', 'multihop', 'named', 'phf' ,'sendmail', 'snmpgetattack', 'snmpguess', 'spy', 'warezclient', 'warezm aster', 'xlock', 'xsnoop', 'worm']

Funkcijom za mapiranje napada prolazimo kroz podatke i svakom primjeru dodjeljujemo po jednu kategoriju kojoj pripada i za trening i za test set:

def map\_attack(attack):

    if attack in dos\_attacks:

        attack\_type = 'dos'

    elif attack in probe\_attacks:

        attack\_type = 'probe'

    elif attack in u2r\_attacks:

        attack\_type = 'u2r'

    elif attack in r2l\_attacks:

        attack\_type = 'r2l'

    else:

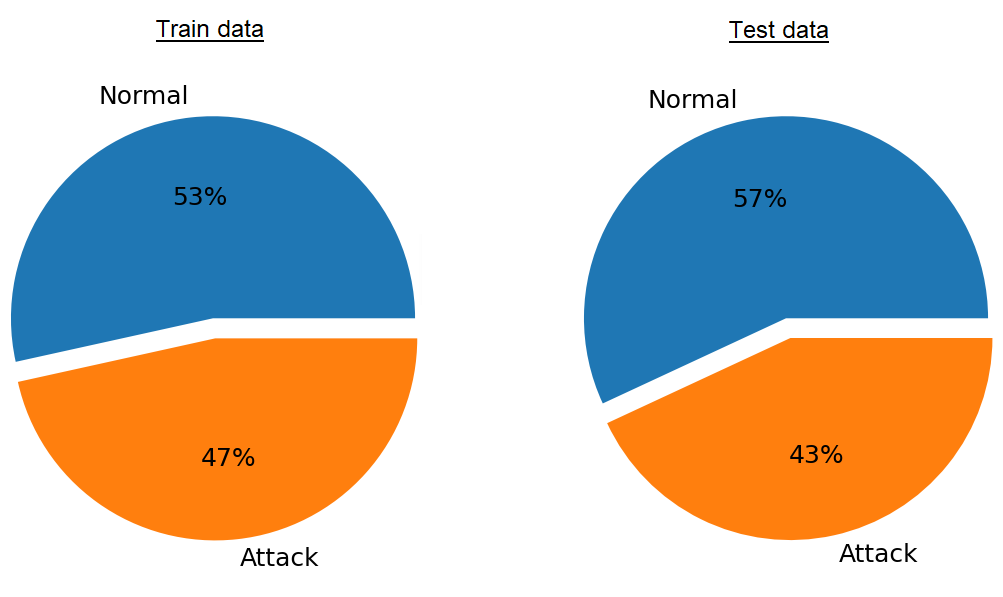
        attack\_type = 'normal'

    return attack\_type

Zbog bolje vizualizacije, nazive za klase unosimo u obliku *stringa*, a kasnije ih prebacujemo u numerički oblik kako bi algoritam mogao raditi s njima.

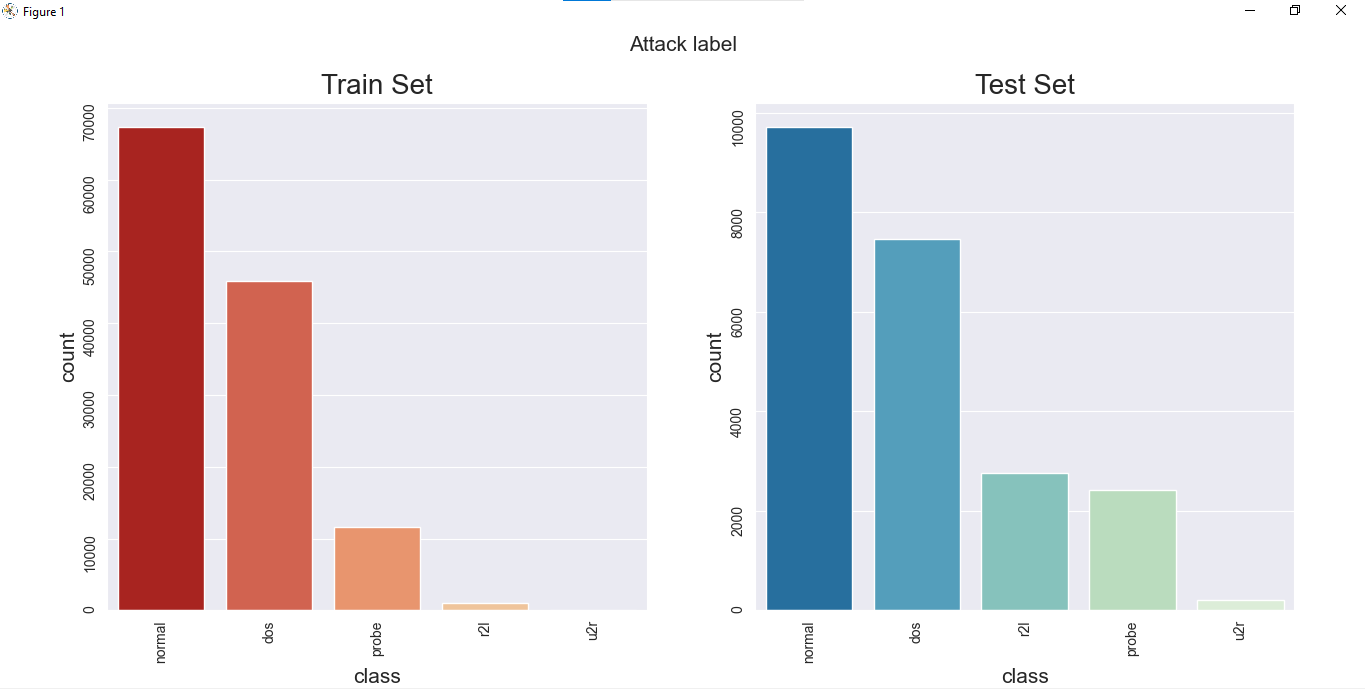
### Vizualizacija

Ono što nas prvenstveno zanima je raspodjela normalnih i sumnjivih veza u trening i test setu, zapravo odnos između njih (Slika 6.4).



Slika 6.4. Odnos između normalnih veza i napada

Iz grafova se vidi da je taj odnos jako sličan za oba seta, ali s obzirom da se radi o *multiclass* klasifikaciji, gdje je normalno povezivanje jedna od klasa, moglo bi doći do pristranosti prema toj klasi. Zbog toga je važno pobliže promotriti klase koje su prisutne u našim setovima (Slika 6.5).



Slika 6.5. Histogrami učestalosti klasa za trening i test set

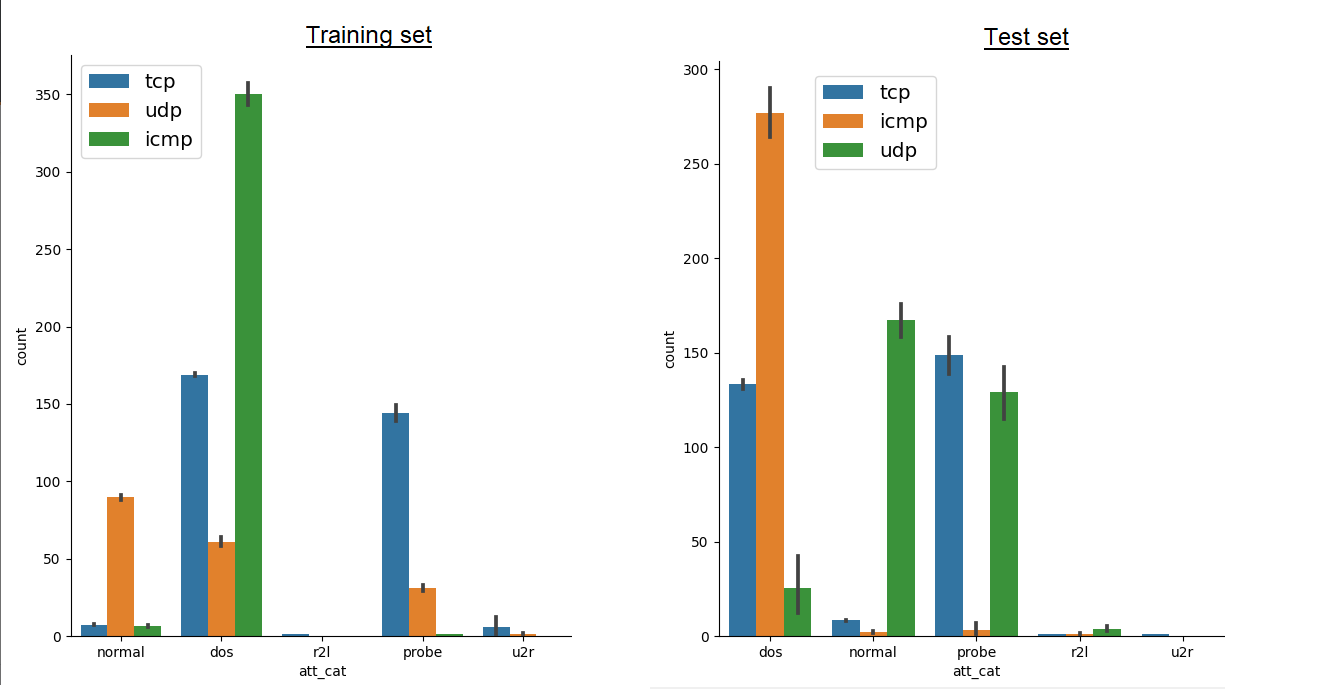
Iz rezultata je vidljivo da su klase prilično dobro raspoređene, odnosno nema nekih prevelikih razlika koje bi utjecale na točno predviđanje algoritma. Ono što se može popraviti jest da u trening setu imamo jako mali broj primjera za klasu *u2r* pa se eventualno neki primjeri mogu prebaciti iz test seta u trening set, ali s obzirom da je svakako ta klasa zastupljena u malom broju i u test setu, može se nastaviti postupak EDA.

Nakon toga potrebno je provjeriti postoji li poveznica između *protocol­­\_type*, *service* i *flag* značajki i samih klasa, odnosno napada i normalnih veza.

Postoje tri vrste protokola koji su prisutni u podacima:

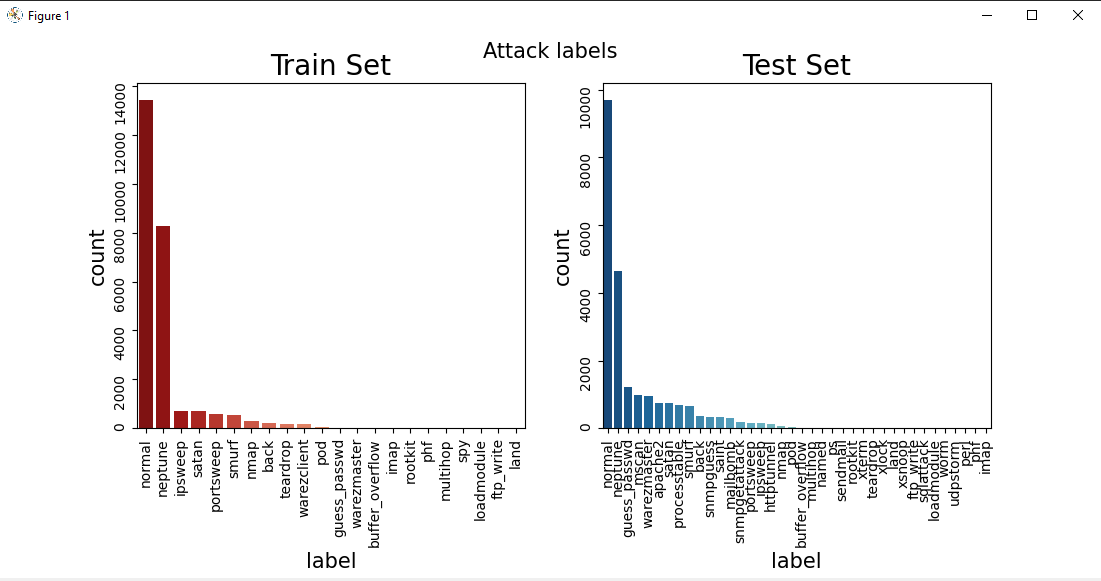
* Transmission Control Protocol (TCP)
* User Datagram Protocol (UDP)
* Internet Control Message Protocol (ICMP)

Prvo provjeravamo kako su raspoređeni protokoli s obzirom na klase što je prikazano na slici 6.6.



Slika 6.6. Zastupljenost protokola po klasama

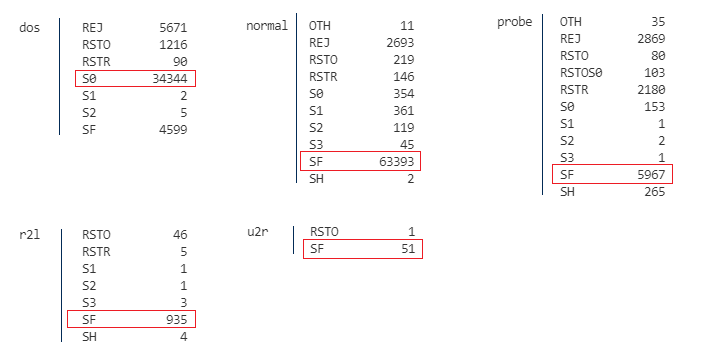
Iz grafa je vidljivo da je normalna veza većinom karakterizirana UDP protokolom, što je jako značajno za samu klasifikaciju. Također, ističe se ICMP protokol koji je karakterističan za *DoS* napade. [30] To se događa zbog učestalosti *DoS* napada koji koriste taj protokol.



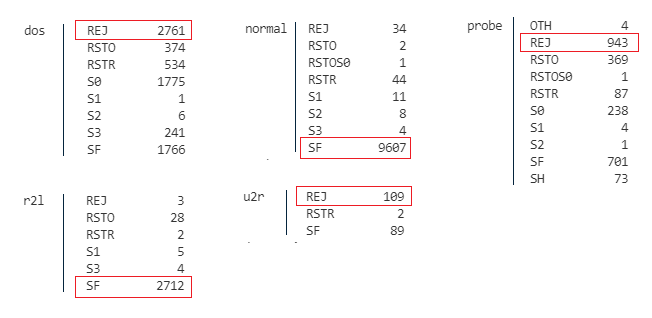
Slika 6.7. Učestalost napada

Npr. napad *smurf*,koji je čest i u trening i u test setu, pripada skupini *DoS* napada, a karakterizira ga slanje ICMP *broadcast* paketa velikom broju poslužitelja (Slika 6.7).

Sada je potrebno provjeriti postoji li veza između značajke *flag* i određenih klasa. Značajkom *flag* definira se status mrežne veze, odnosno radi li se o urednoj vezi ili je došlo do pogreške. Zastupljenost te značajke prikazana je na slikama 6.8 i 6.9.



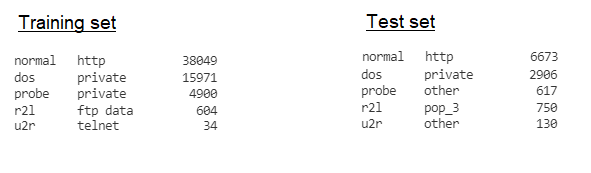
Slika 6.8. Zastupljenost flag značajki po klasama za trening set



Slika 6.9. Zastupljenost flag značajki po klasama za test set

Iz rezultata vidio da je *DoS* klasa u trening setu većinom karakterizirana S0 zastavicom, dok kod test seta većina veza ima zastavicu REJ. Za normalnu vezu, u oba je slučaja riječ o SF zastavici isto kao i kod *r2l* klase. Za *u2r* i *probe* klase najčešće zastavice su SF i REJ. Iz ovoga vidimo da postoji neko pravilo, ali s obzirom da postoje i primjeri koji odstupaju potrebno je uzeti u obzir sve moguće kombinacije.

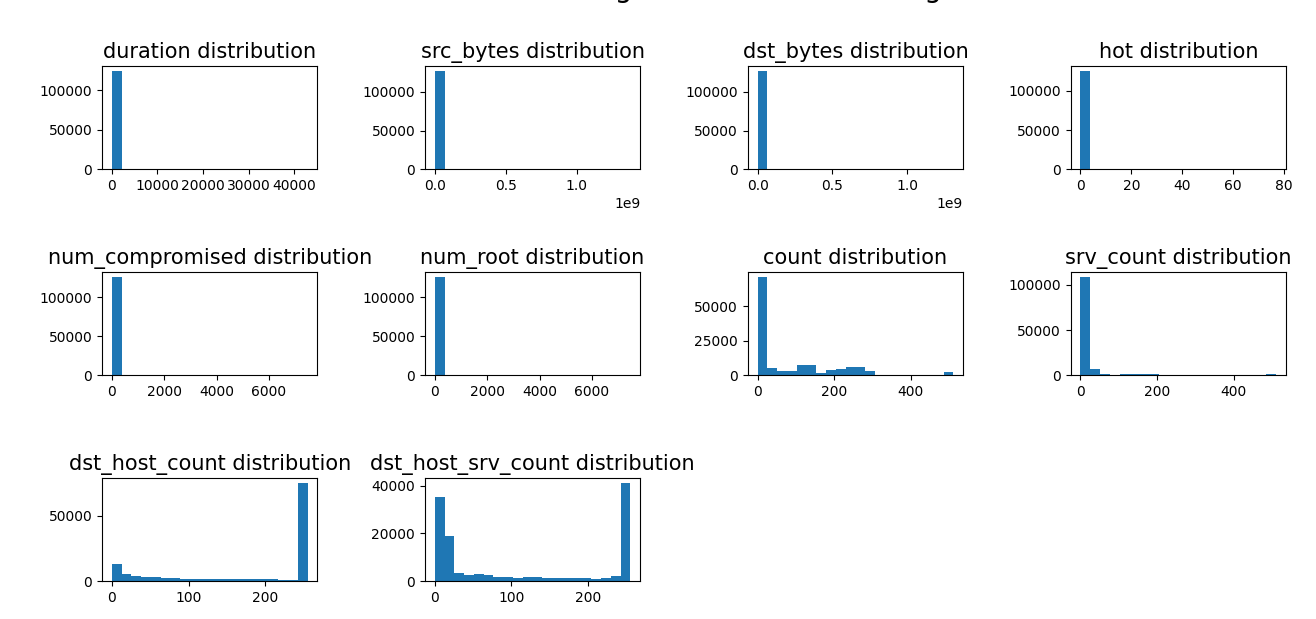
Preostalo je provjeriti i značajku *service* koja označava koja je usluga odredišne mreže korištena. Na slici 6.10 su prikazane one usluge koje se najčešće pojavljuju za određenu klasu.



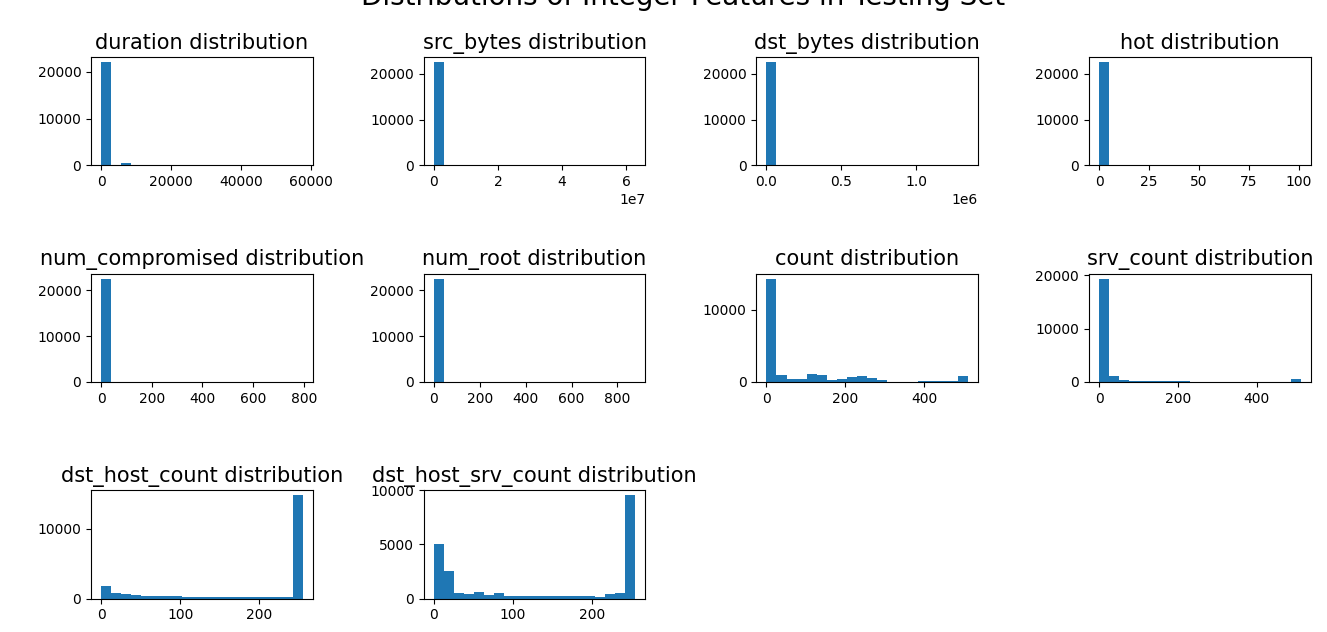
Slika 6.10. Najčešće service značajke po klasama

Normalne veze su, kako je vidljivo iz rezultata, većinom određene *http* uslugom u oba seta. Za *DoS* klasu najveći broj veza karakterizira *private* usluga ponovo u oba seta. Za ostale klase pojavljuju se i oznake *other* što znači da usluga nije definirana što također može biti dobar pokazatelj da je riječ o nekoj vrsti sumnjive veze.

Sljedeća analiza je analiza distribucije *integer* značajki koje imaju različite raspone (Slika 6.11 i Slika 6.12).



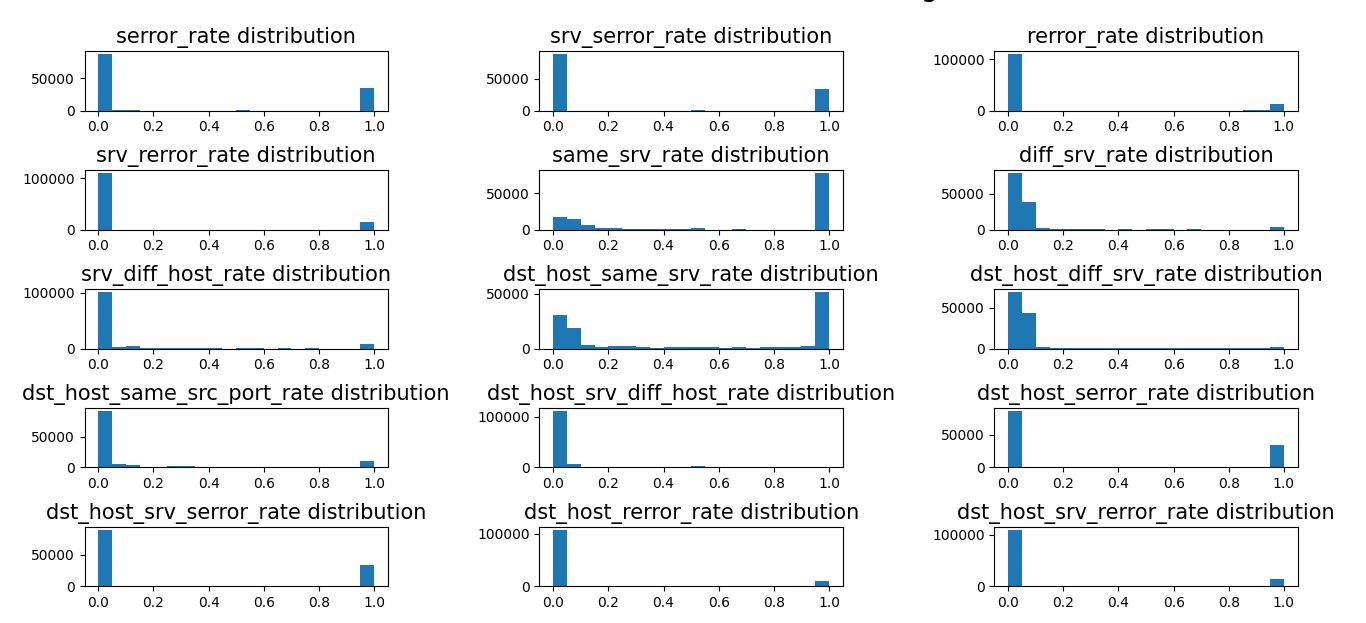
Slika 6.11. Distribucije integer značajki za trening set



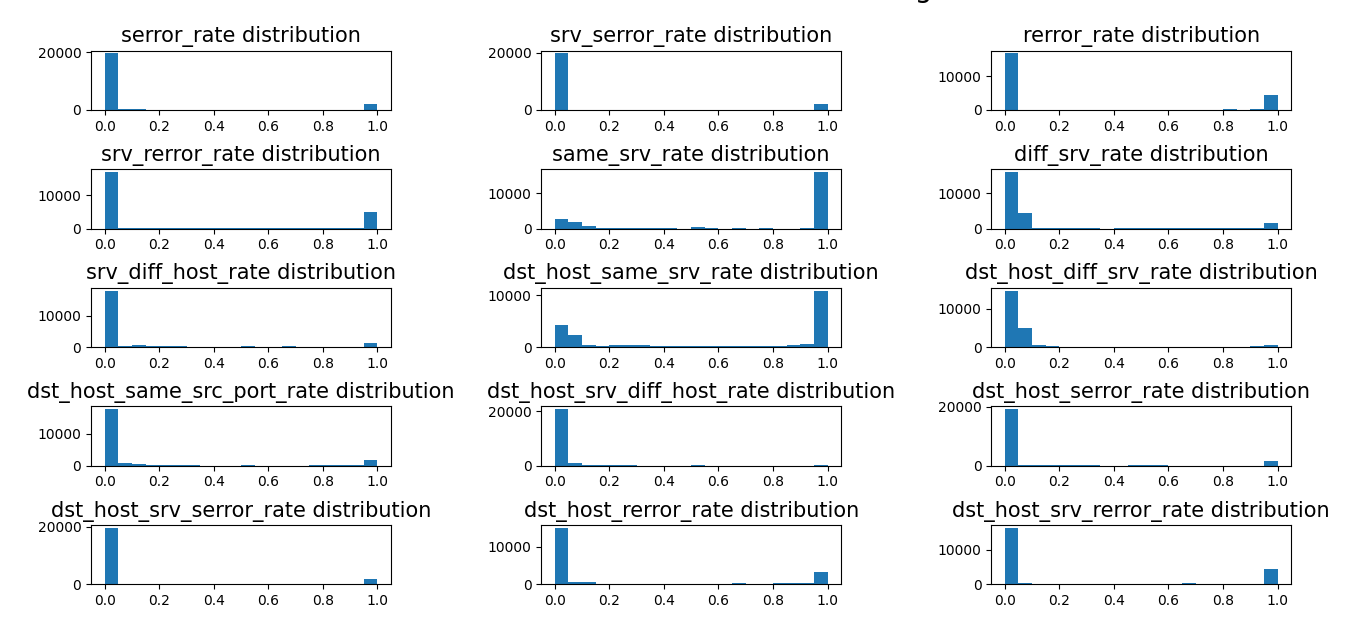
Slika 6.12. Distribucije integer značajki za test set

Iz grafova je vidljivo da u oba seta imamo gotovo jednake distribucije za svaku pojedinu značajku, što odgovara potrebama zadatka, ali problem su distribucije općenito gledano. Grafovi pokazuju koliko se razlikuju distribucije među značajkama i zbog toga se mora provesti normalizacija. Cilj normalizacije promijeniti je vrijednosti numeričkih značajki u skupu podataka kako bi se koristio neki zajednički raspon vrijednosti, bez narušavanja razlika u rasponima ili gubitka podataka.

Provjeravamo i diskretne značajke (Slika 6.13 i Slika 6.14).



Slika 6.13. Distribucije diskretnih značajki za trening set



Slika 6.14. Distribucije diskretnih značajki za test set

Distribucije su jednake, raspon svih vrijednosti je od 0 do 1 pa zaključujemo da su ovi podaci spremni za korištenje i nije potrebno provoditi nikakve postupke nad njima. Ovim korakom, postupak EDA je gotov.

## *Preprocessing i feature engineering*

*Preprocessing* je jako važan, nekada i najvažniji, korak u postupku strojnog učenja. [26] Cilj je pročistiti podatke, eliminirati sve ono nebitno što predstavlja višak te ostaviti samo podatke koji su pouzdani i kvalitetni, odnosno ostaju oni podaci od kojih će algoritam imati koristi pri učenju.

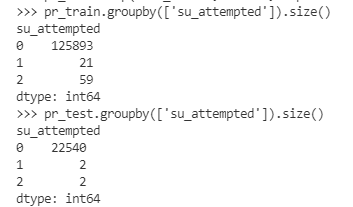
Postupak *preprocessinga* kreće od rješavanja problema vezanih za značajke. Kako smo prethodno definirali, *num\_outbound\_cmds* ima samo vrijednost nula pa nema nikakvo značenje za sam algoritam i tu značajku odbacujemo:

pr\_train.drop('num\_outbound\_cmds', axis=1, inplace=True)

pr\_test.drop('num\_outbound\_cmds', axis=1, inplace=True)

Također odbacujemo i značajku *att\_label* kojom je bila imenovana svaka pojedina mrežna veza jer je ona *string* vrijednost koja ne treba algoritmu pri klasifikaciji, s obzirom da su definirane pripadne klase za svaki primjer.

Nakon toga, potrebno je riješiti problem vrijednosti 2 kod značajke *su\_attempted*. Ta se značajka odnosi na to je li korištena naredba „su root“ (vrijednost 1) ili nije (vrijednost 0). Stoga zaključujemo da je vrijednost 2 pogreška (Slika 6.15).



Slika 6.15. Vrijednosti koje može imati značajka su\_attempted

Budući da se vrijednosti 1 i 2 javljaju u malom broju primjera za razliku od vrijednosti 0, možemo koristiti pretpostavku da je:

Ovakva pretpostavka ne bi trebala imati negativan utjecaj na algoritam i ishode klasifikacije jer se radi o malom postotku primjera u odnosu na ukupan broj.

Što se tiče značajki, još je potrebno riješiti problem vezan uz *service*. Kako smo prethodno definirali, testni set ima nekoliko vrijednosti manje od trening seta i potrebno ih je izjednačiti. To radimo na sljedeći način:

train\_service = pr\_train['service'].tolist()

test\_service = pr\_test['service'].tolist()

difference = list(set(train\_service) - set(test\_service))

string = 'service\_'

difference = [string + x for x in difference]

Kao ispis dobijemo 6 *service* značajki koje nedostaju u testnom setu. One su prikazane na slici 6.16.



Slika 6.16. Razlika service značajki u trening setu u odnosu na test set

Dodajemo ih u testni set kao prazne stupce:

for col in difference:

    pr\_test[col] = 0

### Normalizacija

Prethodno je definirano da normalizacija omogućava stvaranje zajedničkog raspona za vrijednosti svih značajki, uz uvjet da se ne mijenja njihov odnos i da ne dolazi do gubitaka. Postoje različite metode za provedbu normalizacije, a za potrebe ovog rada korišten je *MinMaxScaler*. On transformira značajke tako što ih skalira, odnosno prilagođava i pretvara svaku značajku pojedinačno u zadani raspon. [31] Definira se izrazom (6.1)

|  |  |  |
| --- | --- | --- |
|  |  | (6.1) |

Za dobru primjenu *MinMaxScalera* važno je:

* *MinMaxScaler* postavlja se korištenjem podataka iz trening seta što znači da se koriste trening primjeri za procjenu *min* i *max* vrijednosti te se koristi *fit*() funkcija
* Dobiveni zajednički raspon primjenjuje se na podatke funkcijom *transform*()
* Dalje se dobiveni raspon može primjenjivati na neke nove podatke

U ovom radu *MinMaxScaler* korišten je na sljedeći način:

1. Uključuju se potrebni moduli i biblioteke

from sklearn.preprocessing import MinMaxScaler

1. Definiraju se podaci koje je potrebno normalizirati  
    norm\_cols = ['duration', 'src\_bytes', 'dst\_bytes', 'hot', 'num\_compromised', 'num\_root','num\_file\_creations', 'count', 'srv\_count', 'dst\_host\_count', 'dst\_host\_srv\_count']
2. Definira se MinMaxScaler() te se koristi metoda fit\_transform(X[, y]) koja prvo postavlja podatke pa ih transformira

mms = MinMaxScaler()

pr\_train[norm\_cols] = pd.DataFrame(

    mms.fit\_transform(pr\_train[norm\_cols]), columns=norm\_cols

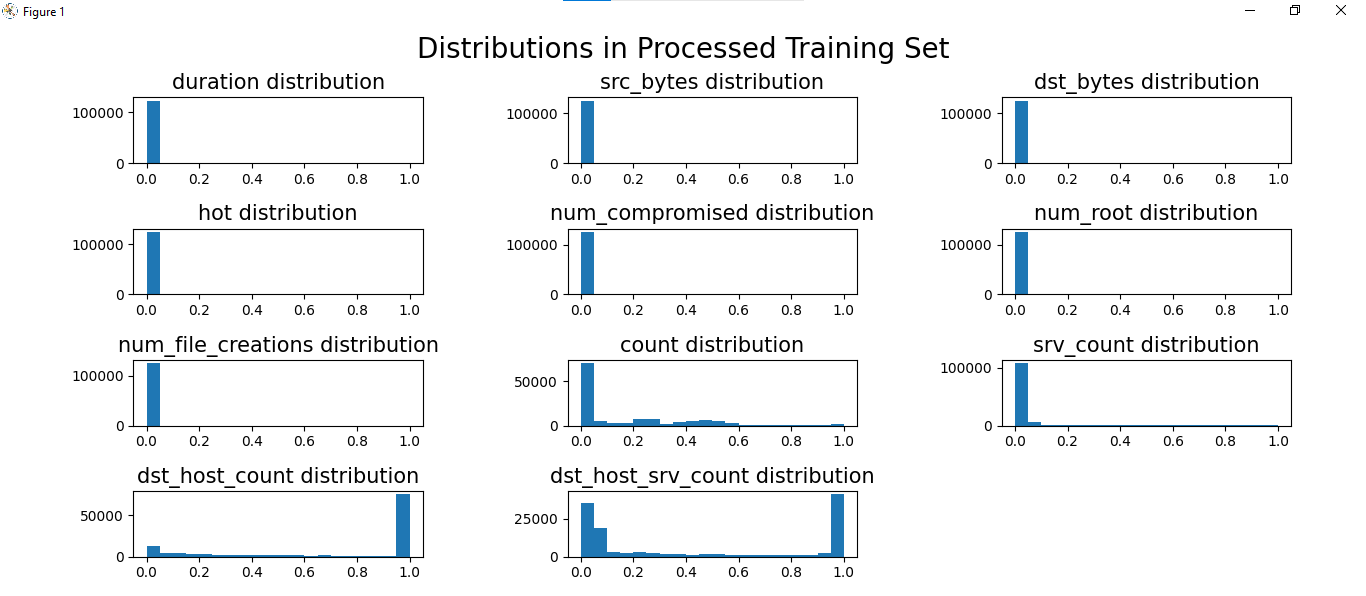
)

pr\_test[norm\_cols] = pd.DataFrame(

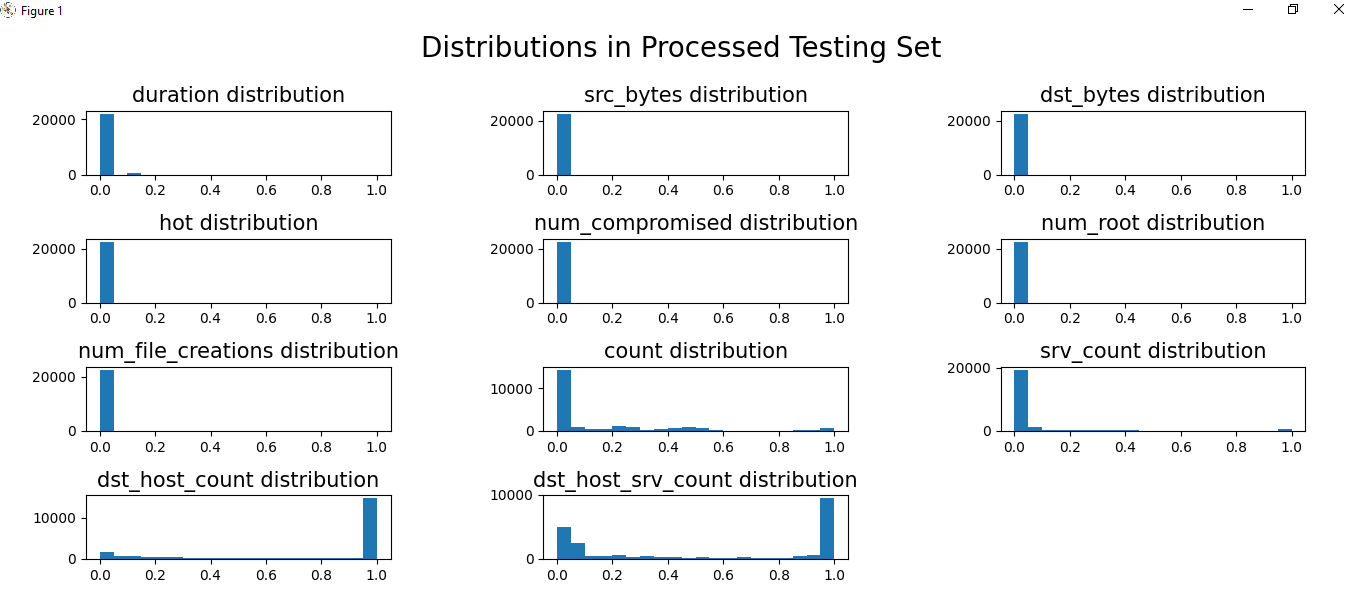
    mms.fit\_transform(pr\_test[norm\_cols]), columns=norm\_cols

)

Nakon izvršavanja definiranog koda, podaci se prikazuju histogramom kako bi se usporedili rasponi i vrijednosti koje smo prethodno imali Slika 6.17. Distribucija u trening setu nakon processinga (Slika 6.17 i Slika 6.18).



Slika 6.17. Distribucija u trening setu nakon processinga



Slika 6.18. Distribucija u test setu nakon processinga

Vidljivo je da su distribucije sada u jednakim rasponima za sve značajke, a distribucije po pojedinoj značajki zadržale su isti oblik. Normalizacija je uspješno provedena. Raspon vrijednosti sada je [0,1], što je *defaultna* vrijednost, ali on se može mijenjati argumentom *feature\_range* po potrebi.

### Label-encoding i One-hot-encoding

Kako smo već definirali, u setovima podataka postoje primjeri s kategoričkim značajkama (značajke koje nisu numeričke). One stvaraju problem algoritmima koji su korištene u ovom radu jer takvi algoritmi zahtijevaju da ulazne i izlazne vrijednosti budu numeričke. Taj se problem rješava korištenjem *label-encodinga* i *one-hot-encodinga* zajedno. [32] Prvenstveno ih je potrebno uključiti preko *sklearn* biblioteke.

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

Sam *label-encoding*  je jako jednostavan postupak koji omogućava da se svaka vrijednost značajke pretvori u numerički oblik.

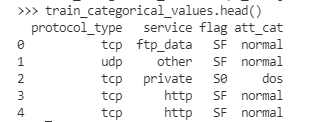
Postupak je sljedeći:

1. Definira se koje kategoričke značajke je potrebno pretvoriti u numerički oblik, primjeri takvih značajki prikazani su na slici 6.19.

categorical\_columns = ['protocol\_type', 'service', 'flag','att\_cat']

train\_categorical\_values = pr\_train[categorical\_columns]

test\_categorical\_values = pr\_test[categorical\_columns]



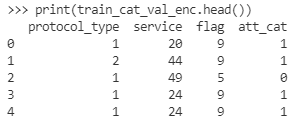
Slika 6.19. Kategoričke značajke i njihove vrijednosti

1. Poziva se *LabelEncoder()* s metodom *fit\_transform(y)* koja postavlja enkoder i kao rezultat vraća enkodirane vrijednosti (Slika 6.20).

train\_cat\_val\_enc = train\_categorical\_values.apply(

    LabelEncoder().fit\_transform)

test\_cat\_val\_enc = test\_categorical\_values.apply(LabelEncoder().fit\_transform)



Slika 6.20. Kategoričke značajke i njihove vrijednosti nakon primjene label-encodinga

Iako je ovaj postupak vrlo jasan i jednostavan, ima i određenu manu. Naime, *label-encoder* numerira značajke redoslijedom kojim se one javljaju (taj broj nema nikakve veze s njihovim značenjem ni odnosom među značajkama) pa određeni algoritam te vrijednosti može krivo interpretirati kao neku vrstu hijerarhije ili redoslijeda. Da bi se taj problem izbjegao koristi se *one-hot-encoder*.

*One-hot-encoder* radi na malo drugačiji način. Za svaku kategoričku vrijednost stvara se poseban stupac i dodjeljuje se vrijednost 0 ili 1 (odgovaralo bi izrazima *true* ili *false*). Koraci za opisane podatke su sljedeći:

1. Za svaku kategoričku značajku definira se string koji predstavlja ime stupca, kao primjer uzmimo značajku *att\_cat* koja označava tražene klase (to su dos, normal, probe, r2l, u2r)

categ\_un = sorted(pr\_train.att\_cat.unique()) #pronađi sve jedinstvene klase

string\_cat = 'cat\_' #dodaj prefiks "cat\_"

categ\_str = [string\_cat + x for x in categ\_un] #npr. "cat\_dos"

1. Nazive stupaca za sve značajke upisujemo u zajedničku listu

col\_name\_dummy = protocol\_str + service\_str + flag\_str + categ\_str

1. Definira se *OneHotEncoder(),* primjenjuje metoda *fit\_transform(X, y=None)* koja postavlja enkoder te transformira vrijednosti *X* (Slika 6.21).

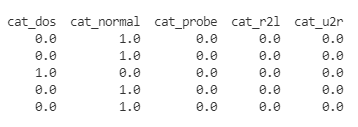
enc = OneHotEncoder()

train\_cat\_val\_hotenc = enc.fit\_transform(train\_cat\_val\_enc)

train\_cat\_data = pd.DataFrame(train\_cat\_val\_hotenc.toarray(), columns=col\_name\_dummy)

test\_cat\_val\_hotenc = enc.fit\_transform(test\_cat\_val\_enc)

test\_cat\_data = pd.DataFrame(test\_cat\_val\_hotenc.toarray(), columns=test\_col\_name\_dummy)



Slika 6.21. Klase nakon one-hot-encodinga

Kao što je vidljivo iz rezultata, kategorije su ponovo jednake kao na slici 6.20, ali u drugom obliku. Ovakav pristup je fleksibilniji jer omogućava enkodiranje proizvoljnog broja kategoričkih značajki te izbor oznake stupca pomoću prefiksa što olakšava daljnje analize.

## Stvaranje i treniranje modela

Za potrebe ovog rada korištena su i uspoređena dva modela: model logističke regresije te neuronska mreža. Za oba modela provedena je *multiclass* klasifikacija te su korišteni isti trening i test setovi podataka.

### Logistička regresija

Prvi korak kod stvaranja bilo kojeg modela je odvajanje primjera sa značajkama od sami klasa, odnosno podjela na *X* i *y* (enkodiranje provodimo kako bi vrijednosti za *y* bile u klasičnom obliku).

X = new\_train.drop(

    ['cat\_dos', 'cat\_normal', 'cat\_probe', 'cat\_r2l', 'cat\_u2r'], axis=1)

y = new\_train[['cat\_dos', 'cat\_normal',

               'cat\_probe', 'cat\_r2l', 'cat\_u2r']]

encoder = LabelEncoder()

y = y.idxmax(1)

encoder.fit\_transform(y)

y = encoder.transform(y)

X\_t = new\_test.drop(

    ['cat\_dos', 'cat\_normal', 'cat\_probe', 'cat\_r2l', 'cat\_u2r'], axis=1)

y\_t = new\_test[['cat\_dos', 'cat\_normal',

               'cat\_probe', 'cat\_r2l', 'cat\_u2r']]

y\_t = y\_t.idxmax(1)

encoder.fit(y\_t)

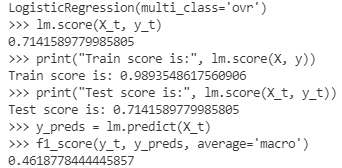
y\_t = encoder.transform(y\_t)

Za logističku regresiju model se stvara korištenjem *sklearn* biblioteke koja omogućava *multiclass* klasifikaciju. Kreće se od jednostavnog modela s *defaultnim* parametrima koji se onda mijenjaju prema potrebama zadatka, jedino što je zadano je korištenje *one-vs-reset* metode kako bi klasifikacija bila *multiclass*. Početni model zadan je kao:

lm = linear\_model.LogisticRegression(

    multi\_class='ovr')

lm.fit(X, y)



Slika 6.22. Rezultat prvog modela logističke regresije

Iz rezultata (Slika 6.22) je vidljivo da je točnost za trening set 98%, dok je za test set 71%, uz F1 *score* koji iznosi 0.46. Iz dobivenih postotaka zaključujemo da je riječ o prenaučenosti, što znači da je potrebno neke parametre postaviti drugačije za bolji rezultat. Neka od osnovnih rješenja su normalizacija i skaliranje te regularizacija.

Model s najboljim rezultatom definiran je kao:

lm = linear\_model.LogisticRegression(

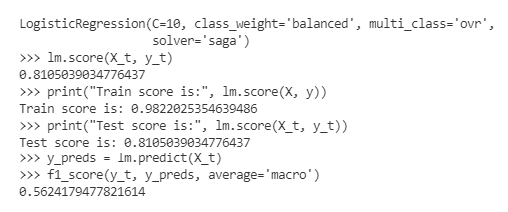
    multi\_class='ovr', class\_weight='balanced', solver='saga', C=10)

lm.fit(X\_lr, y\_lr)

U ovaj model dodan je parametar *class\_weight='balanced'* (*defaultna* vrijednost je *none*) kojim se omogućava da raspored klasa bude balansiran, odnosno da se učestalost klase uzima u obzir. Koristi se vrijednost *y* da bi se automatski ažurirala „težina“ klase kao obrnuto proporcionalna u odnosu na frekvenciju klase u danim podacima. [33]  
Zadani algoritam za ovakav model je Limited-memory Broyden–Fletcher–Goldfarb–Shanno (označava se kao LBFGS) koji nije brz ako se radi s većim setovima podataka. Zbog toga se umjesto njega koristi Stochastic Average Gradient descent (SAG), odnosno njegova ekstenzija *saga*. Ovaj algoritam koristi varijaciju gradijentnog spusta i inkrementalno agregiranog gradijenta koja koristi nasumični uzorak prethodnih gradijentnih vrijednosti. *Saga* omogućava i regularizaciju te je dobar algoritam za veće skupove podataka. Omogućava brže treniranje od *sag* algoritma.

Zadnji parametar je parametar C koji predstavlja inverz snage regularizacije i mora biti pozitivna *float* vrijednost. *Default* vrijednost je 1, a u ovom slučaju više odgovara slabija regularizacija od zadane pa je stoga postavljeno C=10.

Rezultat za ovakav model sada je: točnost za trening set 98%, dok je za test set 81%, uz F1 *score* koji iznosi 0.56, što je zadovoljavajuće za ovaj zadatak (Slika 6.23).



Slika 6.23. Rezultat novog modela logističke regresije

### Neuronska mreža

Za stvaranje modela neuronske mreže, ponovo koristimo odvajanje podataka na *X* i *y*, ovaj put bez dodatnog enkodiranja:

x\_mclss = new\_train.drop(

    ['cat\_dos', 'cat\_normal', 'cat\_probe', 'cat\_r2l', 'cat\_u2r'], axis=1)

y\_mclss = new\_train[['cat\_dos', 'cat\_normal','cat\_probe', 'cat\_r2l', 'cat\_u2r']]

x\_tmc = new\_test.drop(

    ['cat\_dos', 'cat\_normal', 'cat\_probe', 'cat\_r2l', 'cat\_u2r'], axis=1)

y\_tmc = new\_test[['cat\_dos', 'cat\_normal', 'cat\_probe', 'cat\_r2l', 'cat\_u2r']]

Za stvaranje neuronske mreže instaliran je Tensorflow 2.4.1 čiji *backend* koristi biblioteka Keras koja je korištena za stvaranje modela. [34] Kao i za prethodni model, kreće se od jednostavnije varijante (koristi se *sequential* model koji omogućava stvaranje jednostavnog stoga slojeva):

model = Sequential()

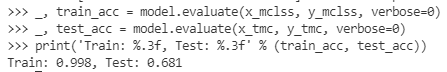
model.add(Dense(32, input\_shape=(x\_mclss.shape[1],), activation='relu'))

model.add(Dense(5, activation='softmax'))

model.summary()

model.compile( loss='categorical\_crossentropy', optimizer='adam',metrics=['accuracy'])

history = model.fit(x\_mclss, y\_mclss, epochs=15, shuffle=True, validation\_split=0.15, verbose=1)



Slika 6.24. Rezultat prvog modela neuronske mreže

U ovom slučaju model s jednim skrivenim slojem daje rezultat, odnosno točnost za trening set od visokih 99%, dok je za test set samo 68% (Slika 6.24). Ovakav rezultat ponovo bi mogao značiti da je riječ o prenaučenosti i ponovo je potrebno uvoditi promjene i druge parametre.

Model s najboljim rezultatom definiran je kao:

model = Sequential()

model.add(Dense(32, input\_shape=(X.shape[1],), kernel\_regularizer=l2(

    0.001), bias\_regularizer=l2(0.001), activation='sigmoid'))

model.add(keras.layers.Dropout(0.3))

model.add(Dense(16, activation='sigmoid'))

model.add(keras.layers.Dropout(0.3))

model.add(Dense(5, activation='softmax'))

model.summary()

model.compile( loss='categorical\_crossentropy', optimizer=Adam(learning\_rate=0.001, beta\_1=0.99, beta\_2=0.999, amsgrad=True),metrics=['accuracy'])

es = keras.callbacks.EarlyStopping(monitor='val\_loss',mode='min',patience=4,restore\_best\_weights=True)

history = model.fit(X, dummy\_y, callbacks=[es], epochs=15, batch\_size=60, shuffle=True, validation\_split=0.15, verbose=1)

Uz dodatne slojeve, u novi je model dodano i nekoliko metoda za sprječavanje prenaučenosti. [35] Prvenstveno, korištena je regularizacija kroz regularizatore koji omogućuju penalizaciju za parametre slojeva ili za aktivnost slojeva tijekom optimizacije. Regularizacija zapravo smanjuje prenaučenost tako što modificira funkciju koštanja.

Korištena su dva regularizatora:

* *kernel\_regularizer*: regularizaciju koja se primjenjuje na *kernel* sloja
* *bias\_regularizer*: regularizacija na pristranost sloja

Uz to, za regularizaciju se koriste i *dropout* slojevi koji za razliku od regularizatora, modificiraju samu neuronsku mrežu. *Dropout* sloj nasumično ispušta neurone iz neuronske mreže tijekom treninga u svakoj iteraciji. Kad ispustimo različite skupove neurona, to je ekvivalentno treniranju različitih neuronskih mreža. Različite mreže imaju različite prenaučenosti pa će učinak *dropout* sloja biti da smanjuje prenaučenost. Dobro je koristiti male vrijednosti za *dropout* oko 20%-50%.

Aktivacijska funkcija za skrivene slojeve promijenjena je u *sigmoid* funkciju koju smo prethodno opisali zbog toga što više odgovara danom zadatku, a u zadnjem sloju u oba modela korištena je *softmax* funkcija koja se najčešće i koristi u zadnjem sloju za klasifikaciju jer se njezin rezultat može interpretirati kao distribucija vjerojatnosti.

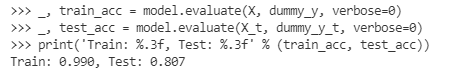
Optimizator je ostao isti, samo su dodane vrijednosti koje su promijenjene u odnosu na *defaultne*:

* *learning\_rate* povećan s 0.001 na 0.01
* *beta\_1* – eksponencijalna brzina propadanja povećana s 0.9 na 0.99
* *amsgrad* *True* umjesto *False* (omogućava primjenu AMSGrad varijante algoritma)

U novom modelu korišten je i *Early stopping* koji zaustavlja postupak treniranja kada se promatrana metrika prestane poboljšavati kroz iteracije, u našem slučaju nakon 4 iteracije.

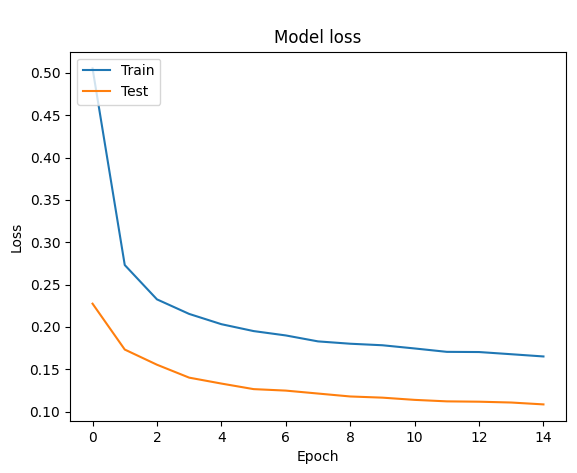
Povećan je i *batch\_size* kojim se definira broj primjera koji propagiraju kroz mrežu.

Rezultat koji dobijemo novim modelom je:



Slika 6.25. Rezultat novog modela neuronske mreže

Točnost za trening set i dalje je 99%, ali sada za test set imamo 80% što je zadovoljavajuće za potrebe rada (Slika 6.25). Ako iscrtamo krivulju učenja za setove vidimo da ne spada u područja ni prenaučenosti ni podnaučenosti (Slika 6.26).



Slika 6.26. Krivulja učenja za setove podataka

## Usporedba modela

Oba modela na kraju daju slične rezultate, odnosno točnost. Što se tiče primjene, logistička regresija je jednostavnija zbog manje parametara, ali je i sporija. Neuronska mreža može postići sve ono što može i logistička regresija pa čak i brže, ali potrebno je dobro razumjeti parametre s kojima se radi i prilagoditi ih na način koji odgovara potrebama podataka. Uz to, neuronska mreža je računski skuplja pa je uvijek dobro krenuti od jednostavnijih algoritama.

# ZAKLJUČAK

# LITERATURA

1. <https://en.wikipedia.org/wiki/Artificial_intelligence>, s Interneta (18.6.2021.)

1. <https://cdn.auraquantic.com/wp-content/uploads/2020/04/Artificial-Intelligence-img-1024x1024-1.jpg>, „Artificial Intelligence Technologies and their categories“, Alba Fernández, Aura Quantic

1. <https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/>, s Interneta (14.6.2021.)
2. https://www.guru99.com/images/1/030819\_1026\_SupervisedM2.png

1. <https://www.fer.unizg.hr/_download/repository/StrojnoUcenje.pdf>, s Interneta (14.6.2021.)

1. <https://www.fer.unizg.hr/_download/repository/SU-2017-02-OsnovniKoncepti.pdf>, s Interneta (12.6.2021.)

1. <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>, s Interneta (13.6.2021.)

1. <https://mukulrathi.co.uk/demystifying-deep-learning/debug-neural-network-learning/>, „Debugging the Learning Curve“, 2.rujna 2018., Mukul Rathi
2. Ethem Alpaydin, „Introduction to Machine Learning, Second Edition“, 2010 Massachusetts Institute of Technology

1. <https://en.wikipedia.org/wiki/Supervised_learning>, s Interneta (12.6.2021.)

1. <https://www.moogsoft.com/blog/aiops/understanding-machine-learning-aiops-part-2>, 1.kolovoza 2017., Robert Harper

1. <https://en.wikipedia.org/wiki/Cluster_analysis>, s Interneta (13.6.2021.)

1. <https://en.wikipedia.org/wiki/Semi-supervised_learning>, s Interneta (15.6.2021.)

1. <https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/> , s Interneta (15.6.2021.)

1. <https://perfectial.com/wp-content/uploads/2018/07/img2-7.jpg>, „Reinforcement Learning Applications: A Brief Guide on How to Get Business Value from RL“, Rostyslav Demush

1. <https://www.fer.unizg.hr/_download/repository/StrojnoUcenje.pdf>, s Interneta (14.6.2021.)

1. <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/> , s Interneta (15.6.2021.)

1. <https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b>, s Interneta (15.6.2021.)

1. <https://4.bp.blogspot.com/-sCcOrQsTH9Q/XG1yv7mhERI/AAAAAAAAAJI/aEj6Jf1lookERHqPQS_Y6Q9bxBcTV7TIwCLcBGAs/s1600/multiclass-multilabel.png>, „Evaluation Metrics for Multi-Label Classification“, 15.ožujka , Pritish Jadhav

1. [https://miro.medium.com/max/970/1\*Xu7B5y9gp0iL5ooBj7LtWw.png](https://miro.medium.com/max/970/1*Xu7B5y9gp0iL5ooBj7LtWw.png), „Topic DL01: Activation functions and its Types in Artifical Neural network“, abhigoku10, 27. ožujka 2018.

1. [https://miro.medium.com/max/600/1\*iNPHcCxIvcm7RwkRaMTx1g.jpeg](https://miro.medium.com/max/600/1*iNPHcCxIvcm7RwkRaMTx1g.jpeg), „Data Science the smart way: Regression — Part II“, Shamim Ahmed, 9.rujna 2020.
2. <http://uc-r.github.io/public/images/analytics/gbm/learning_rate_comparison.png>
3. <https://s18798.pcdn.co/yungjurick/wp-content/uploads/sites/12997/2020/03/1_eBMwpBBboAXgqsawwOKkPw.png>

1. <https://en.wikipedia.org/wiki/Kaggle>, s Interneta (23.6.2021.)

1. <https://www.unb.ca/cic/datasets/nsl.html>, s Interneta (23.6.2021.)

1. <https://numpy.org/doc/stable/user/whatisnumpy.html>, s Interneta (23.6.2021.)

1. <https://en.wikipedia.org/wiki/Pandas_(software)> , s Interneta (23.6.2021.)

1. <https://towardsdatascience.com/exploratory-data-analysis-eda-techniques-for-kaggle-competition-beginners-be4237c3c3a9>, s Interneta (25.6.2021.)

1. <https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657>, s Interneta (25.6.2021.)

1. <https://us-cert.cisa.gov/ncas/tips/ST04-015>, s Interneta (27.6.2021.)

1. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>, s Interneta (29.6.2021.)

1. <https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd>, s Interneta (29.6.2021.)

1. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>, s Interneta (30.6.2021.)
2. <https://keras.io/guides/sequential_model/>, s Interneta (01.07.2021.)

1. <https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html>, s Interneta (01.07.2021.)

# POPIS OZNAKA I KRATICA

DoS Denial of Service

EDA Exploratory Data Analysis

FN false negative

FP false positive

ICMP Internet Control Message Protocol

LBFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno

ML Machine learning

R2L Remote to Local

SAG Stochastic Average Gradient descent

TN true negative

TP true positive

TCP Transmission Control Protocol

UI Umjetna inteligencija

UDP User Datagram Protocol

U2R User to Root

ANN artificial neural network

BP backpropagation

SMR softmax regression

PCA Principal Component Analysis

# SAŽETAK

Zadatak ovog diplomskog rada bio je pregled područja strojnog učenja, odnosno pregled osnovnih koraka strojnog učenja te njegovih alata i metoda. Također, pronaći i proučiti gotovi skup podataka te nad njim provesti strojno učenje korištenjem usvojenih metoda.

# KLJUČNE RIJEČI

# SUMMARY

# TITLE

# KEYWORDS