Evaluacija automobila primjenom umjetne inteligencije

Ilijanić, Luka

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:308978

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

EVALUACIJA AUTOMOBILA PRIMJENOM UMJETNE INTELIGENCIJE

Rijeka, ožujak 2023.

Luka Ilijanić
0069080019

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

EVALUACIJA AUTOMOBILA PRIMJENOM UMJETNE INTELIGENCIJE

Mentor: Prof. dr. sc. Zlatan Car

Rijeka, ožujak 2023.

Luka Ilijanić
0069080019

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Primjena umjetne inteligencije**
Grana: **2.03.06 automatizacija i robotika**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Luka Ilijanić (0069080019)**
Studij: Diplomski sveučilišni studij elektrotehnike
Modul: Automatika

Zadatak: **Evaluacija automobila primjenom umjetne inteligencije/Car evaluation based on artificial intelligence method**

Opis zadatka:

Napraviti pregled postojećih istraživanja i primjene algoritama umjetne inteligencije korištenih u evaluaciji automobila. Primijeniti nekoliko različitih algoritama umjetne inteligencije za evaluaciju automobila. Algoritme trenirati I testirati na javno dostupnom skupu podataka, te izvršiti usporedbu dobivenih rezultata. Za izradu algoritama koristiti Python programski jezik.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Prof. dr. sc. Zlatan Car

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Viktor Sučić

IZJAVA

Ja, Luka Ilijanić, JMBAG: 0069080019, student Tehničkog fakulteta u Rijeci smjera diplomskog sveučilišnog studija elektrotehnike, izjavljujem da sam diplomski rad "Evaluacija automobila primjenom umjetne inteligencije" samostalno izradio sukladno članku 8. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci uz pomoć navedene literature i mentora prof. dr. sc. Zlatana Cara te da su svi dijelovi rada rezultat isključivo mojega vlastitog rada.

Rijeka, ožujak 2023. Luka Ilijanić

_____

ZAHVALA

Zahvaljujem prof. dr. sc. Zlatanu Caru što mi je omogućio pisanje diplomskog rada pod njegovim mentorstvom. Također, zahvaljujem se i asistentu dr. sc. Nikoli Anđeliću na ukazanoj pomoći i savjetima tokom procesa pisanja diplomskog rada. Ponajviše se zahvaljujem mojoj obitelji koja je vjerovala u mene i pružala mi podršku kroz čitav period studiranja.

Sadržaj

1. UVOD

Razvojem interneta, bez kojeg je u današnje vrijeme gotovo nemoguće raditi, kroz zadnjih nekoliko godina raste popularnost umjetne inteligencije. Umjetna inteligencija može se definirati kao sposobnost stroja/sustava da u pravom trenutku povuče pravi potez, odnosno donese ispravnu odluku. Riječ je o alatu koji nastoji čovjeku olakšati put do ostvarenja željenog cilja zbog čega je primjenjen u području medicine, ekonomije, proizvodnje, itd. Iako zbog ubrzanog razvoja postoji zabrinutost da će umjetna inteligencija doseći razinu u kojoj će preuzeti većinu ljudskog rada, takav scenarij još nije primjetan, premda je njena uporaba zapravo postala dio svakodnovnog života, ponajviše zbog korištenja mobilnih uređaja (prepoznavanje lica, glasovni asistenti, automatsko ispravljanje teksta, itd.). Glavnu podvrstu umjetne inteligencije čini strojno učenje zahvaljujući kojem stroj, umjesto rješavanja problema iz ograničenog područja za koje je isprogramiran, na temelju određenog algoritma uči svojstva iz dobivenih podataka i na taj način postaje sposoban samostalno rješavati i probleme s kojima se ranije nije susreo.

Jedna od uspješnih primjena umjetne inteligencije je korištenje algoritama umjetne inteligencije za procjenu vrijednosti automobila. Zbog kompleksne strukture automobila prednost čovjeka prilikom procjene vrijednosti još uvjek je u tome što ima pristup automobilu i može detaljnim pregledom uočiti moguće nedostatke, međutim u tom će slučaju odluka ovisiti o razini njegova iskustva i mogućeg subjektivnog mišljenja. Ipak, glavni nedostatak je u tome što čovjek u slučaju većeg broja automobila ne može ni približno brzo dati procjene na osnovu saznanja njihovih karakteristika, zbog čega danas postoji mnoštvo aplikacija koje temeljem raznih skupova podataka u trenutku pružaju grubu procjenu zadanog vozila.

Cilj ovog rada jest učiti (trenirati) i testirati različite algoritme strojnog učenja, kao što su logistička regresija, algoritam k-najbližih susjeda, višeslojni perceptron i stablo odlučivanja, na skupu podataka evaluacije automobila. Osim toga glavna namjera je usporediti rezultate osnovnih metrika strojnog učenja dobivene uporabom zadanih vrijednosti hiperparametara s rezultatima dobivenim metodom nasumičnog pretraživanja vrijednosti hiperparametara čija svrha jest postizanje veće klasifikacijske točnosti. Također, cilj je ostvariti i što stabilniju procjenu izvedbe pojedinog modela korištenjem metode k-struke unakrsne validacije te otkriti može li se njihovom međusobnom kombinacijom postići još bolji rezultat.

Korišteni skup podataka vrednuje automobile na osnovi njihove cijene, troška održavanja, broja vrata, broja osoba koje stanu u automobil, veličine prtljažnika i razine sigurnosti. U drugom poglavlju prikazan je kratak osvrt na literaturu, odnosno radove koji su u svom istraživanju također koristili navedeni skup. U trećem poglavlju rada opisan je skup podataka na način da se prikazuje pojedinačan utjecaj ulaznih varijabli na izlaznu varijablu. Zaključci o njihovom međusobnom odnosu doneseni su na temelju grafičkih prikaza, statističkog testa i korelacijske analize. Kroz četvrto poglavlje razrađen je proces pripreme skupa podataka za učenje i testiranje algoritama. Pritom su definirane metrike na osnovu čijih rezultata se ispituje kvaliteta pojedinog algoritma, tj. modela. U petom poglavlju detaljno su opisani korišteni modeli strojnog učenja. Za svaki model realizirana je petlja unutar koje algoritam nasumično odabire vrijednosti hiperparametara i nastoji ostvariti što bolji rezultat za svaku metriku. U šestom poglavlju opisan je postupak izvođenja metode unakrsne validacije kojom se osigurava stabilnija procjena izvedbe modela koji se nastoji trenirati. Uz to predstavljen je i način na koji je moguće kombinirati algoritme strojnog učenja u cilju postizanja boljih rezultata. Na samom kraju, kroz sedmo poglavlje iznesen je kratak komentar na usporedbu rezultata dobivenih u ovome radu s rezultatima dobivenim u navedenoj literaturi. Za potrebe izvođenja ovog rada koristi se Python programski jezik.

2. PREGLED LITERATURE

U istraživanjima [1,2] korišten je isti skup podataka za evaluaciju automobila koji je kreirao Marko Bohanec 1997. godine. Autori su pritom usporedili različite klasifikacijske algoritme te zaključke o kvaliteti donijeli na temelju razine točnosti i brzine izvođenja pojedinog algoritma. U [1] testirana je učinkovitost algoritma naivnog Bayesa (eng. Naive Bayes) i višeslojnog perceptrona (eng. Multi-Layer Perceptron (MLP)), koji se uz algoritam stabla odlučivanja (eng. Decision Tree) također pojavljuju i u istraživanju [2]. U svakom radu autori analiziraju učinkovitost algoritama u slučaju podjele skupa podataka na skupove za treniranje i testiranje za tri različita omjera; 90:10, 66:34 i 50:50. Uz to, izvedena je i metoda unakrsne validacije, točnije 10-fold unakrsna validacija, a procjena izvedbe pojedinog modela definirana je na osnovu rezultata metrike točnosti kojeg je algoritam ostvario na testnom skupu. Kako postotak skupa za trening raste tako se na testnom skupu ostvaruju sve bolji rezultati točnosti, koji zahvaljujući MLP klasifikatoru u istraživanju [1] maksimalno dosežu vrijednost od 94.79%, dok u istraživanju [2] za svaki klasifikator vrijedi maksimalna točnost od 93.06%. Maksimalna vrijednost točnosti dobivena 10-fold unakrsnom validacijom u istraživanju [1] također je ostvarena MLP klasifikatorom te iznosi 94.09%, dok je za isti klasifikator u istraživanju [2] postignuta točnost od 93.51%, odnosno 93.22% za klasifikator stabla odlučivanja. Iako postoji razlika u dobivenim rezultatima doneseni zaključci su isti u oba rada, a to je da algoritam višeslojnog perceptrona pruža najvišu točnost, no zbog svoje kompleksnosti zahtjeva više vremena za izradu i testiranje modela u usporedbi sa stablom odlučivanja i naivnim Bayesovim modelom.

3. STATISTIČKA ANALIZA PODATAKA

Statističkom analizom podataka nastoji se iz korištenog skupa izvući zaključci o njegovoj strukturi te međusobnoj ovisnosti promatranih varijabli. Vrijednosti pojedinačnih varijabli, kao i parova *ulazna varijabla – izlazna varijabla* vizualizirat će se pomoću stupčastih grafova nakon čega će se provesti statistički test radi ispitivanja povezanosti između ulaznih varijabli i izlazne varijable. Na kraju, kako bi se i numerički predstavila međusobna ovisnost dviju varijabli te potvrdile tvrdnje donesene na temelju stupčastih grafova i statističkog testa, odradit će se korelacijska analiza.

3.1. Pregled podataka

Za početak potrebno je učitati korišteni skup podataka te dobiti uvid o kojoj količini i kakvoj vrsti podataka je riječ. Pritom će poslužiti funkcija `dtypes`, dok se za samo učitavanje, budući da je riječ o CSV datoteci, koristi naredba `read_csv` koja dolazi unutar knjižnice Pandas, čija je odlika brzo, fleksibilno i jednostavano analiziranje i manipuliranje podacima. Dobiveni izlazi prikazani su tablicama 3.1. i 3.2.

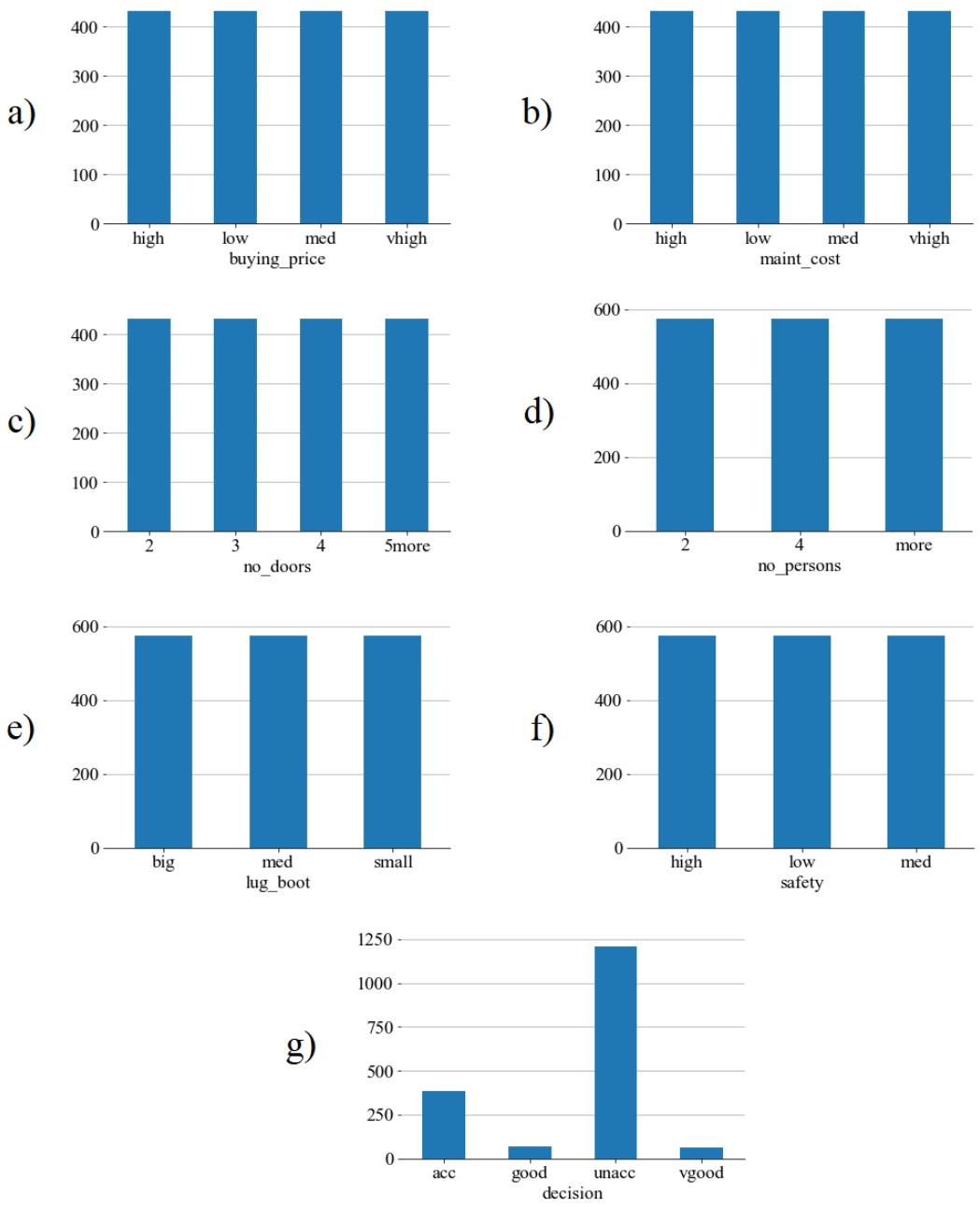
Tablica 3.1. Struktura skupa podataka "car evaluation"

	buying_price	**maint_cost**	**no_doors**	**no_persons**	**lug_boot**	**safety**	**decision**
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

Tablica 3.2. Vrste podatka za "car evaluation" skup podataka

Varijabla	**Tip podatka**
buying_price	object
maint_cost	object
no_doors	object
no_persons	object
lug_boot	object
safety	object
decision	object

Iz tablica 3.1 i 3.2 vidi se kako skup podataka sadržava 1728 redaka i 7 stupaca (varijabli) te su svi podaci tipa *object*, odnosno riječ je o kategoričkim varijablama. Kao izlazna vrijabla koristi se varijabla *decision*. Iako se već iz tablice 3.1. vidi kako neke od varijabli posjeduju brojčanu vrijednost, ti brojevi zapravo nemaju isto značenje kao kod numeričkih varijabli te je nemoguće nad njima vršiti matematičke operacije. Kako bi se prebrojale i prikazale vrijednosti koje pojedine varijable mogu poprimiti, definira se funkcija koja će ispisati stupčaste grafove (dodatak A). Kao rezultat dobivaju se grafovi prikazani na slici 3.1. Slika 3.1 prikazuje raspodjelu vrijednosti za pojedinu varijablu. Horizontalna os prikazuje vrijednosti koje se uspoređuju, a vertikalna os u ovom slučaju predstavlja brojčano stanje.

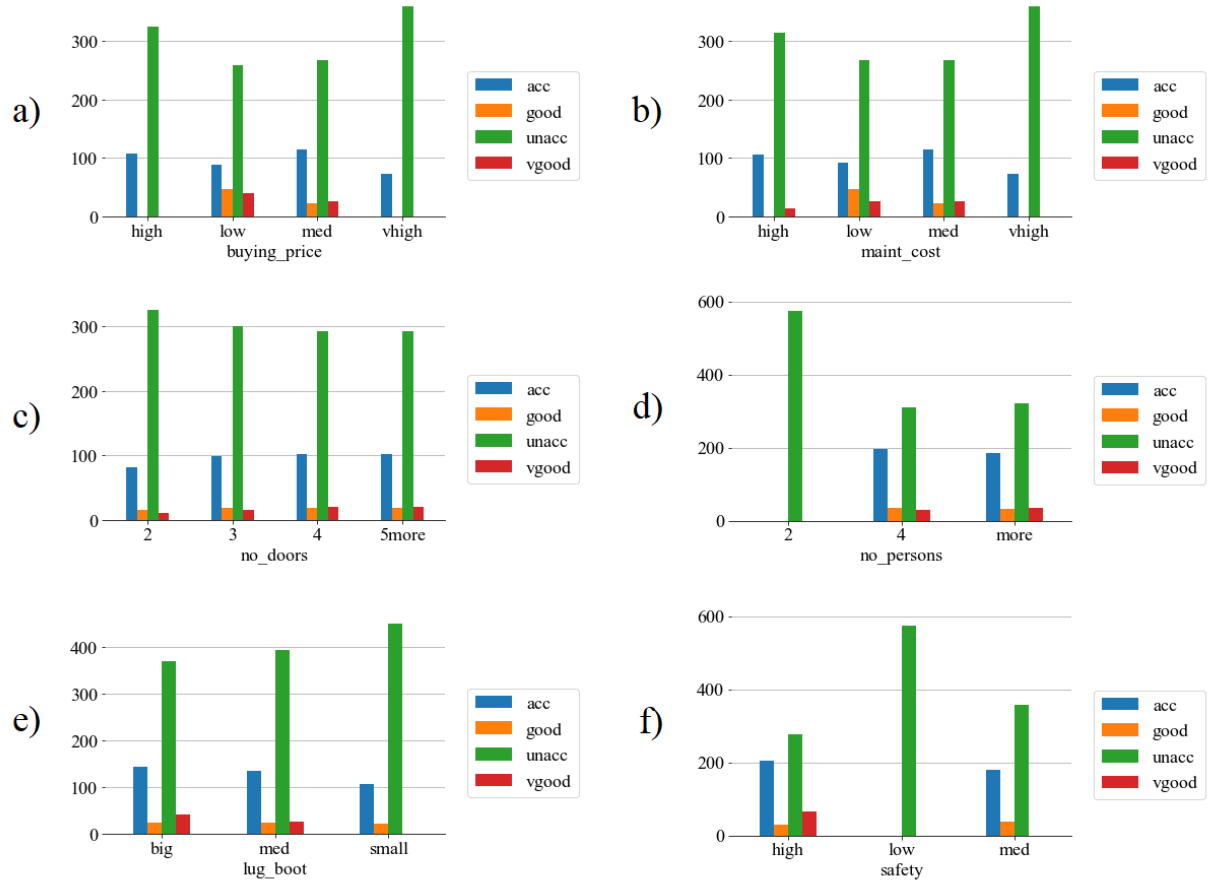


Slika 3.1. Grafički prikaz raspodjele vrijednosti varijable a) buying_price, b) maint_cost, c) no_doors, d) no_persons, e) lug_boot, f) safety i g) decision

Vidi se kako svaka ulazna varijabla (slika 3.3.a, b, c, d, e i f) ima jednoliko raspoređene vrijednosti (jednolika razdioba), što s druge strane nije slučaj kod izlazne varijable (slika 3.3.g) kod koje je daleko najveća vjerojatnost da će poprimiti vrijednost *unacc*, odnosno da će se većina automobila smatrati neprihvatljivim.

Sličnim postupkom realizirana je i funkcija kojom će se grafički vizualizirati utjecaj svake vrijednosti ulazne varijable na varijablu *decision* (dodatak A). Na taj način dobiva se okviran

uvid u njihovu međusobnu zavisnost. Dobiveni grafovi prikazani su na slici 3.2. Slika 3.2 prikazuje utjecaj vrijednosti pojedinih ulaznih varijabli na izlaznu varijablu.



Slika 3.2. Grafički prikaz utjecaja vrijednosti varijable a) buying_price, b) maint_cost, c) no_doors, d) no_persons, e) lug_boot i f) safety na varijablu decision

U slučaju utjecaja varijable *buying_price* na varijablu *decision* (slika 3.2.a) vidi se da bolje vrednuju automobili sa srednjom i niskom kupovnom cijenom, što je i očekivano. Isto tako promatrajući varijablu *maint_cost* (slika 3.2.b), odnosno trošak održavanja, uočava se kako bolje prolaze automobili sa srednjim i niskim troškom održavanja. Nešto drukčija situacija je ukoliko se promatra utjecaj broja vrata te broja osoba koji stanu u automobil (slika 3.2.c i 3.2.d). Broj vrata gotovo da nema značajnog utjecaja na odluku, dok se s druge strane svi automobili u koje ne stanu barem 4 osobe smatraju neprihvatljivim. Kada se gleda veličina prtljažnika (slika 3.2.e) najlošije prolaze oni s malim, dok se podjednako vrednuju oni s srednjim i velikim prtljažnim prostorom. Na kraju sigurnost (slika 3.2.f), kao jedna od najbitnijih značajki prilikom odabira automobila, ako je na visokoj razini može lako donijeti dobru ocjenu, no ukoliko je razina sigurnosti niska automobil će se smatrati potpuno neprihvatljivim.

Ovakav ishod gotovo da je i očekivan, budući da većina ljudi vrednuje automobile na sličan ili možda čak isti način. Dobiveni grafovi poslužili su samo kao gruba procjena ovisnosti izlazne o ulazim varijablama, dok će precizna vjerojatnost suodnosa biti opisana u nastavku.

3.2. Hi-kvadrat test

Hi-kvadrat (χ^2) test statistički je test koji se koristi za određivanje povezanosti između kategoričkih varijabli analizom odnosa između promatranih i očekivanih vrijednosti. Obično se primjenjuje ako su podaci kvalitativni, odnosno ako se podaci mogu promatrati na temelju određenih karakteristika, atributa, svojstva, itd., što je upravo slučaj kod ovog skupa podataka. Kako bi se provjerila postojanost veze između dvije kategoričke varijable koristi se formula hi – kvadrat testa:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}, \tag{3.1}$$

gdje je:

χ^2 – hi-kvadrat,

O_i – promatrana vrijednost,

E_i – očekivana vrijednost i

n – broj ćelija.

Test se provodi na način da za početak definiraju dvije hipoteze:

- H0 (nul hipoteza): dvije kategoričke varijable su nezavisne i
- H1 (alternativna hipoteza): dvije kategoričke varijable su zavisne.

U ovom slučaju provjerava se postoji li veza između ulaznih i izlazne varijable uz pomoć ranije formiranih tablica (grafova). Za primjer će se uzeti tablica odnosa između varijabli *buying_price* i *decision*. Ona u ovom slučaju predstavlja tablicu promatranih vrijednosti O_i, odnosno izvorni podatak.

Tablica 3.3. Odnos varijabli buying_price i decision (O)

buying_price	acc	good	unacc	vgood	Σ
high	108	0	324	0	432
low	89	46	258	39	432
med	115	23	268	26	432
vhigh	72	0	360	0	432
Σ	384	69	1210	65	1728

Tablica očekivanih vrijednosti $E_{i,}$ tj. tablica s teoretskim rezultatima koji se očekuju prema pravilima vjerojatnosti dobiva se na način da se zbroj retka pomnoži sa zbrojem stupca te se podijeli s ukupnim zbrojem svih redaka, odnosno stupaca. Primjenom tog postupka dobiva se tablica 3.4. Uočava se kako su vrijednosti po stupcima identične, a razlog tomu jest jednolika raspoređenost vrijednosti varijable *buying_price*.

Tablica 3.4. "Očekivan" odnos varijabli buying_price i decision (E)

buying_price	acc	good	unacc	vgood
high	96	17.25	302.5	16.25
low	96	17.25	302.5	16.25
med	96	17.25	302.5	16.25
vhigh	96	17.25	302.5	16.25

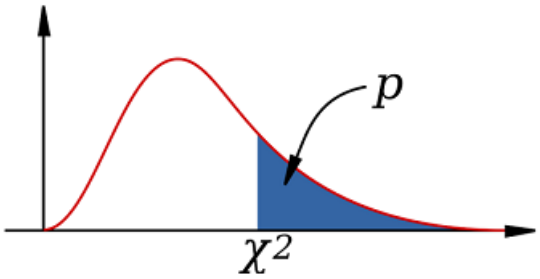
Sada kada su poznate vrijednosti O_i i E_i, jednostavnim matematičkim operacijama dolazi se do vrijednosti $\chi^2 = 189.243$. Poznavajnjem iznosa χ^2 postoje dvije mogućnosti za dokazivanje zašto bi se, odnosno ne bi trebala prihvatiti nul hipoteza. Prvi način se izvodi uz pomoć tablice graničnih vrijednosti koja pokazuje do koje vrijednosti, uz određeni broj stupnjeva slobode, se smatra da je hi-kvadrat još uvijek dovoljno visok da bi se mogla odbaciti nul hipoteza. Drugim riječima, koliko mora najmanje iznositi hi-kvadrat da se odbaci nul hipoteza [3]. Jedan dio tablice graničnih vrijednosti prikazan je tablicom 3.5.

Tablica 3.5. Dio tablice graničnih vrijednosti hi – kvadrata [4]

Br. stupjeva slobode	**Vrijednost razine značajnosti**								
	0.99	**0.95**	**0.90**	**0.75**	**0.50**	**0.25**	**0.10**	**0.05**	**0.01**
1	0.000	0.004	0.016	0.102	0.455	1.32	2.71	3.84	6.63
2	0.020	0.103	0.211	0.575	1.386	2.77	4.61	5.99	9.21
3	0.115	0.352	0.584	1.212	2.366	4.11	6.25	7.81	11.34
4	0.297	0.711	1.064	1.923	3.357	5.39	7.78	9.49	13.28
5	0.554	1.145	1.610	2.675	4.351	6.63	9.24	11.07	15.09
6	0.872	1.635	2.204	3.455	5.348	7.84	10.64	12.59	16.81
7	1.239	2.167	2.833	4.255	6.346	9.04	12.02	14.07	18.48
8	1.647	2.733	3.490	5.071	7.344	10.22	13.36	15.51	20.09
9	2.088	3.325	4.168	5.899	8.343	11.39	14.68	16.92	21.67
10	2.558	3.940	4.865	6.737	9.342	12.55	15.99	18.31	23.21

Granična vrijednost hi-kvadrata može se dobiti uz poznavanje broja stupnjeva slobode – *d*, te vrijednost razine značajnosti – α. Broj stupnjeva slobode računa se na način da se pomnoži broj redaka i stupaca, pri čemu se svaki segment unaprijed umanji za 1. Uobičajen iznos razine značajnosti za donošenje odluke je 5%, što ustvari predstavlja dopuštenu pogrešku. Stoga, za ovaj primjer vrijedi da će se sa 9 stupnjeva slobode te uz $\alpha = 0.05$, za graničnu vrijednost dobiti 16.92. Budući da je χ^2 veći od granične vrijednosti odbacuje se hipoteza H0, što znači da je riječ o zavisnim varijablama.

Drugi način je usporedba p-vrijednosti i definirane razine značajnosti. P-vrijednost daje informaciju o vjerojatnosti pojave podataka pod nultom hipotezom. Vizualno se može pojasniti slikom 3.3.



Slika 3.3. Primjer prikaza p-vrijednosti hi-kvadrat testa [5]

Definiranim brojem stupnjeva slobode određuje se odgovarajuća krivulja distribucije, a poznavanjem vrijednosti χ^2 površina ispod nje zatim se dijeli na dva dijela, nakon čega se promatra desni dio površine (p-vrijednost). Ukoliko je p-vrijednost manja od definirane razine značajnoti (α) odbacuje se nul hipoteza. Budući da je unutar knjižnice SciPy, koja služi za rješavanje različitih znanstvenih i inženjerskih problema, već implementirana funkcija `chi2_contingency` koja na izlazu daje hi-kvadrat i p-vrijednost, na jednostavan način uspjeva se ustanoviti postoji li povezanost između varijabli. Primjenom navedene funkcije na tablicu *buying_price – decision* dolazi se do rezultata p = 5.9281e-36 što je manje od iznosa razine značajnosti, stoga se i u ovom slučaju odbacuje hipoteza H0. Na isti način djeluje se na sve ostale tablice koje su se ranije kreirale te su p-vrijednosti predočene tablicom 3.6.

Tablica 3.6. P-vrijednosti za odgovarajuće ulazne varijable

Ulazna varijabla	**p-vrijednost**
buying_price	5.9281e-36
maint_cost	2.5476e-26
no_doors	0.3202
no_persons	4.04e-77
lug_boot	1.0294e-09
safety	2.3891e-100

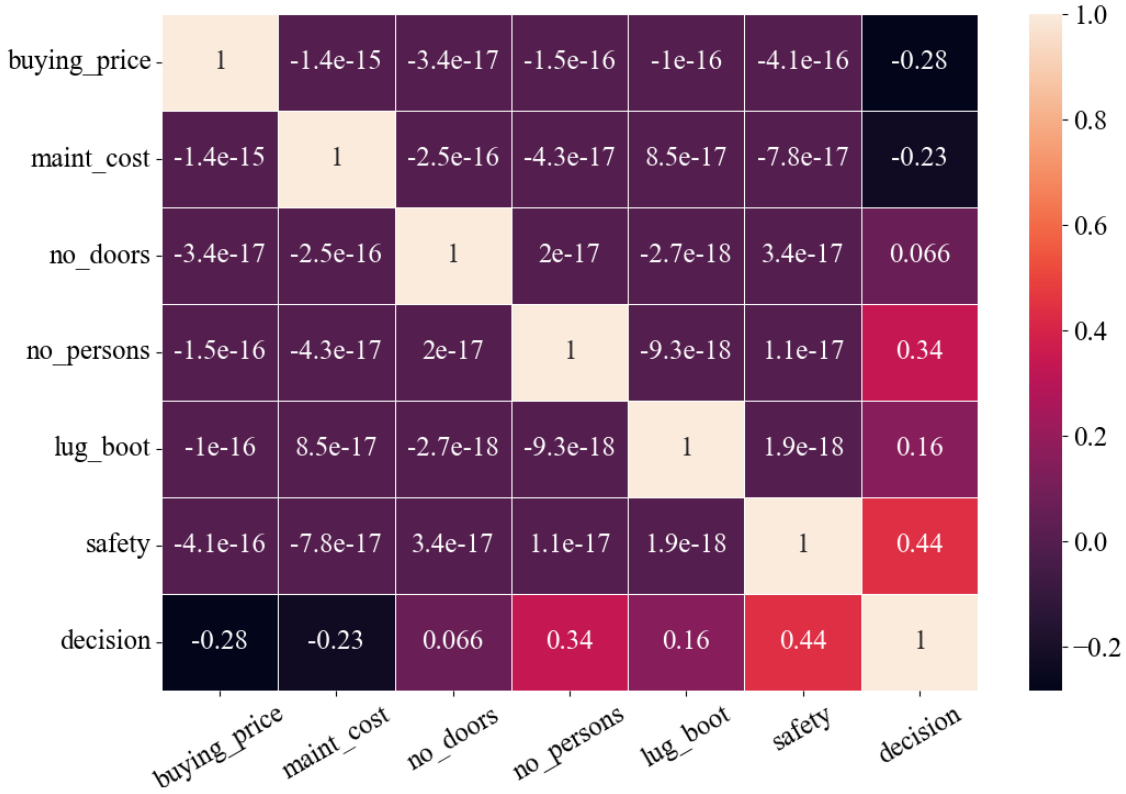
Kao i za slučaj varijable *buying_price,* kod svih ostalih varijabli p-vrijednost je daleko manja od α, osim kod *no_doors* gdje je p = 0.3202, što daje na znanje da se nul hipoteza u ovom slučaju ne odbacuje, tj. da nema značajnog dokaza da su varijable *no_doors i decision* zavisne.

3.3. Korelacijska analiza

Za razliku od hi-kvadrat testa, koji pokazuje vjerojatnost povezanosti između dvije varijable, račun korelacije pokazuje stupanj (visinu) povezanosti [3]. Mjera povezanosti između dviju varijabli određena je koeficijentom korelacije koji može poprimiti bilo koju vrijednost iz intervala [-1, 1]. Vrijednost -1 predstavlja potpunu negativnu povezanost, vrijednost 1 potpunu pozitivnu povezanost dviju varijabli, dok 0 označava nepostojanje povezanosti promatranih varijabli. Drugim riječima, ako je korelacija između dviju varijabli jednaka 1 to

znači da ukoliko vrijednost jedne varijable raste tada će i vrijednost druge varijalbe rasti, i obrnuto. Može se reći kako su najbolji korelacijski rasponi od -1 do -0.5 i od 0.5 do 1, a najlošije korelacijske vrijednosti između -0.5 do 0.5. Najgora moguća korelacijska vrijednost je 0 te u tom slučaju nema nikakve korelacije između varijabli. Općenito, važnost korelacije leži u tome što omogućava da se vrijednost jedne varijable s određenom vjerojatnošću može predvidjeti na osnovi saznanja o vrijednosti druge varijable.

Što se tiče provođenja korelacije unutar programskog paketa Python, ona se može provesti samo ako su kategorički podaci ordinalnog tipa, što znači da vrijednosti varijabli imaju svojstveni poredak. To trenutno ne vrijedi za promatrani slučaj, budući da program nema jasan pogled na vrijednosti *low, med* i *high* kao što ima čovjek. Stoga je potrebno napraviti transformaciju svih varijabli u ordinalne, na način da budu razumljive od strane računala. To se postiže uz pomoć mapiranja (dodatak A). Provedbom korelacijske analize dobivena je tablica prikazana slikom 3.4.



Slika 3.4. Tablica korelacije "car evaluation" skupa podataka

Najveća vrijednost korelacije je kod varijable *safety* što je i očekivano budući da se od ranije zna kako razina sigurnosti ima vrlo značajan utjecaj prilikom vrednovanja automobila,

odnosno da porast razine sigurnosti rezultira boljom ocjenom. Zanemarivi iznosi korelacije između pojedinih ulaznih varijabli pokazuju kako su one međusobno neovisne. Kod iznosa korelacija između pojedinih ulaznih varijabli i izlazne varijable vide se znatno veći iznosi korelacije, što znači da su međusobno ovisne. Negativna korelacija kod varijabli *buying_price* i *maint_cost* znači da će se njihovim povećanjem smanjivati varijabla *decision*, tj. više cijene rezultirati će lošijom ocjenom. Najslabija korelacija očekivano vrijedi kod varijable *no_doors* kod koje je već spomenuto da slabo utječe na procjenu automobila.

Zaključno, uočava se kako se uz pomoć grafičkog prikaza, hi-kvadrat testa i korelacijske analize dolazi do istih zaključaka, stoga se prilikom učenja modela u nastavku ne uzima u obzir varijabla *no_doors*.

4. PRISTUP STROJNOM UČENJU

U ovom dijelu rada opisati će se osnovni pojmovi koje je potrebno znati prilikom implementacije algoritama strojnog učenja. Pojasnit će se način na koji se potrebno pripremiti skup podataka koji će poslužiti za učenje i testiranje algoritma te mjerila na osnovu kojih se dobiva uvid u kvalitetu dobivenih rješenja. Također, obratit će se pozornost na česte probleme koji se mogu pojaviti prilikom treniranja modela strojnog učenja te način na koji ih se može prepoznati i reducirati.

4.1. Strojno učenje i Scikit-learn knjižnica

Scikit-learn je knjižnica čija je glavna značajka u tome što nudi različite alate za odabir modela, prilagodbu modela, evaluaciju modela i slično. Riječ je o knjižnici strojnog učenja koja podržava učenje pod nadzorom i bez nadzora. Učenje pod nadzorom znači da model za treniranje ima dostupne ulazne i željene izlazne podatke, što znači da obradom ulaznih podataka postoji mogućnost usporedbe dobivenog izlaza sa željenim izlazom i na taj je način moguće eliminirati odstupanje, dok kod učenja bez nadzora model ne posjeduje željene izlazne podatke te mora sam donositi odluke koje karakteristike će koristiti da bi grupirao ulazne podatke. U ovom slučaju radi se o učenju s nadzorom. Općenito, problem učenja s nadzorom može biti klasifikacijskog ili regresijskog tipa. Problem klasifikacije odnosi se na proces pronalaženja modela koji pokušava razvrstati podatke u više diskretnih kategorija, što je upravo slučaj kod ovog skupa podataka, dok je problem regresije pronaći funkciju kojom će se što preciznije opisati povezanost varijabli.

Uobičajena praksa u strojnom učenju jest podjela skupa podataka na dva dijela. Jedan od tih skupova naziva se skupom za treniranje (eng. training set), preko kojega se uče određena svojstva, dok se drugi skup naziva skupom za testiranje (eng. testing set) te se na njemu testiraju naučena svojstva. Ta se podjela unutar Python-a postiže korištenjem funkcije `train_test_split` iz Scikit-learn knjižnice. Pritom je potrebno navesti varijable u koje će se pohraniti skup za treniranje, odnosno testiranje te postotak skupa podataka koji će se koristiti za testiranje.

```
X = pd.DataFrame()
X['buying_price'] = coded_data ['buying_price']
X['maint_cost'] = coded_data ['maint_cost']
X['no_persons'] = coded_data ['no_persons']
X['lug_boot'] = coded_data ['lug_boot']
X['safety'] = coded_data ['safety']
y = coded_data ['decision']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=1)
```

U ovom slučaju prije same podjele potrebno je ulazne varijable povezati u jedinstvenu varijablu *X*. To se ostvaruje primjenom Pandas funkcije `DataFrame()` čija je svrha formiranje dvodimenzionalne strukture podataka u kojoj se podaci poravnavaju prema tabličnom obliku. Izlaznim podacima, koji se obično označavaju sa *y*, pridružuje se varijabla *decision*. Što se tiče količine podataka koja će se koristiti u svrhu testiranja navodi se kako će ona iznositi 20%. Želeći pritom da svaki put prilikom izvršavanja skripte isti podaci budu korišteni za treniranje, definira se sjeme koje koristi generator nasumičnih brojeva (*random_state*) i tako spriječi slučajan odabir podataka.

4.2. Evaluacijske metrike

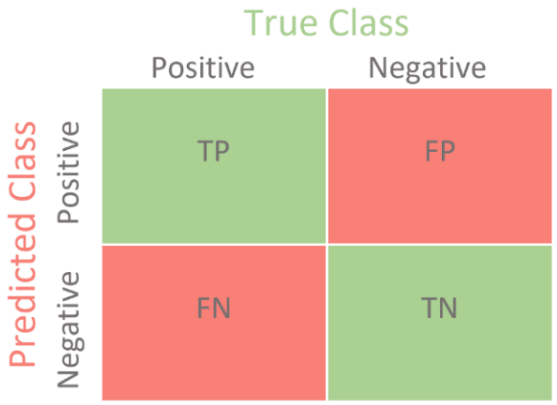
Kako bi se došlo do saznanja koji će od kasnije korištenih algoritama strojnog učenja dati najbolju estimaciju potrebno je napraviti usporedbu na temelju određenih mjernih podataka, tzv. metrika. Nakon primjene nekog od algoritma strojnog učenja, potrebni su određeni alati radi spoznaje koliko je dobro algoritam odradio svoj posao, stoga se za svaki problem strojnog učenja zahtijeva odgovarajući skup metrika za procjenu izvedbe [6].

Za potrebe ovog rada od klasifikacijskih metrika koriste se:

- točnost,
- preciznost,
- opoziv,
- F1 rezultat i
- ROC AUC rezultat.

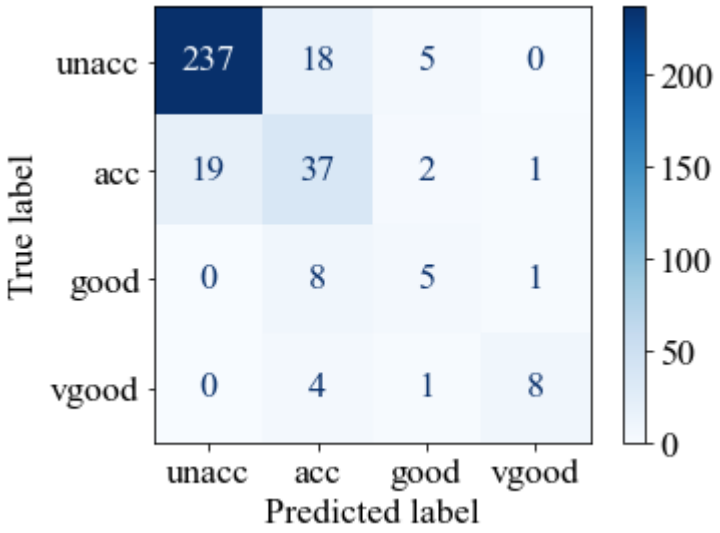
Radi što jasnijeg opisa navedenih metrika za početak se analizira tzv. matrica konfuzije. Matrica konfuzije (eng. confusion matrix) jedna je od najintuitivnijih metrika koja se koristi

za pronalaženje točnosti i ispravnosti algoritma strojnog učenja. Njena glavna upotreba je u problemima klasifikacije gdje izlaz može sadržavati dvije ili više vrsta klasa [6]. Predstavlja tablicu specfičnog rasporeda koja pruža uvid u pojedinačan odnos predviđenih i istinitih klasa. Obično su po stupcima raspoređene predviđene klase, dok su istinite klase raspoređene po retcima. Koristi se kao vizualna podloga na temelju koje se izračunavaju ostale metrike. Matrica konfuzije za slučaj binarne klasifikacije prikazana slikom 4.1.



Slika 4.1. Primjer matrice konfuzije za binarnu klasifikaciju [7]

Točnost (eng. accuracy) je jedna od najčešće korištenih klasifikacijskih metrika. Definirana je kao broj točno predviđenih klasa (TP+TN) podjeljen s ukupnim brojem predviđanja (TP+TN+FP+FN). Iako najjednostavniji i najčešće korišteni pokazatelj za mjerenje učinka klasifikatora, točnost nije uvijek dobra metrika, osobito kada su podaci neuravnoteženi. Osnovni problem je u tome što kada je jedna od klasa dominantna, postiže se visoka točnost ukoliko se većinu vremena predviđa ta ista klasa [8]. Upravo ovaj skup podataka zbog značajne razlike u vrijednostima izlazne varijable primjer je jednog takvog skupa, stoga iznos točnosti neće biti od velikog značaja tokom kasnije analize. Na slici 4.2. prikazana matrica konfuzije promatranog višeklasnog skupa podataka dobivena primjenom jednog od kasnije korištenih algoritama strojnog učenja.



Slika 4.2. Primjer matrice konfuzije "car evaluation" skupa podataka

Iz slike 4.2 jasno se da uočiti kako je u slučaju klase *unacc* većina instanci ispravno klasificirana, dok je kod ostalih klasa znatno manji broj ispravno dodijeljenih klasa, pogotovo kod klase *good*. Naravno, takav ishod je i očekivan s obzirom da se algoritam zbog daleko najveće pripadnosti najbolje istrenirao na klasi *unacc*. Iz primjera se lako može izračunati kako točnost u ovom slučaju iznosi 0.829, odnosno 82.9%.

Slijedeće dvije često korištene metrike su preciznost i opoziv. U slučaju binarne klasifikacije preciznost (eng. precision) je definirana kao broj točno predviđenih pozitivnih klasa (TP) u odnosu na broj predviđanja pozitivnih klasa (TP+FP), dok opoziv (eng. recall) označava broj točno predviđenih pozitivnih klasa (TP) u odnosu na stvaran broj pozitivnih klasa (TP+FN). Stoga, prilikom analiziranja preciznosti teži se ka sigurnosti u stvaran broj pozitivno predviđenih klasa, dok je kod opoziva težnja k tomu da broj lažno negativnih klasa bude što manji. Njihovom kombinacijom dolazi se do sljedeće popularne metrike, a to je F1 rezultat.

F1 rezultat (eng. F1 score), poznata i kao F1 mjera, predstavlja harmonijsku srednju vrijednost preciznosti i opoziva te je dana formulom:

$$F1 = 2 * \frac{precision * recall}{precision + recall}. \tag{4.1}$$

Kao i točnost, F1 rezultat često je korišten u klasifikacijskim problemima, međutim njegova vrijednost bit će od veće koristi ukoliko se barata s neuravnoteženim podacima, tj. kada je

prisutna velika razlika u pripadnosti određenoj klasi. Za razliku od binarne, kod višeklasne klasifikacije F1 rezultat, kao i prethodne dvije metrike, određuje se za svaku klasu zasebno. Unutar programskog paketa Python rezultate tih metrika najlakše je prikazati kroz tzv. klasifikacijski izvještaj korištenjem naredbe `classification_report`. Klasifikacijski izvještaj za matricu sa slike 4.2. prikazan je tablicom 4.1.

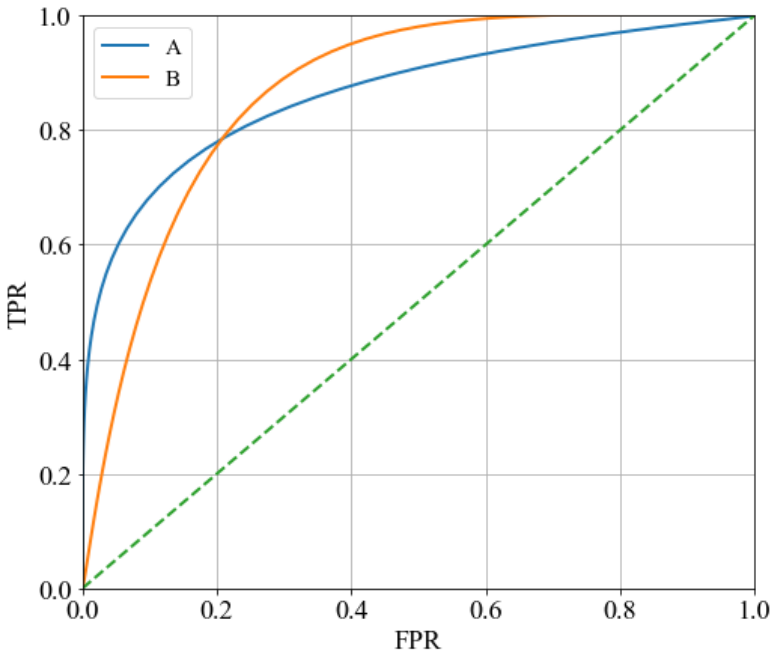
Tablica 4.1. Primjer klasifikacijskog izvještaja "car evaluation" skupa podataka

	precision	**recall**	**f1-score**	**support**
unacc	0.93	0.91	0.92	260
acc	0.55	0.63	0.59	59
good	0.38	0.36	0.37	14
vgood	0.80	0.62	0.70	13
accuracy	/	/	0.83	346
macro avg	0.67	0.63	0.64	346
weighted avg	0.84	0.83	0.83	346

Iz tablice 4.1 se vidi kako rezultati izvještaja odgovaraju zaključcima donesenim prilikom promatranja matrice sa slike 4.2. Međutim, umjesto višestrukih rezultata za svaku metriku, bilo bi bolje izračunati njihov prosjek kako bi se dobio jedan rezultat za opisivanje ukupne kvalitete. Taj se rezultat kod višeklasnih skupova podataka može dobiti računanjem makro prosjeka (eng. macro average) ili pak težinskog prosjeka (eng. weighted average). Razlika u rezultatima je što se makro prosjek izračunava korištenjem aritmetičke sredine svih rezultata, dok je težinski prosjek jednak sumi svih rezultata, pri čemu se rezultat za pojedinu klasu množi s udjelom te klase u izlaznoj varijabli, što je u tablici klasifikacijskog izvještaja navedeno pod stavkom "support". Prativši ideju da trenirani algoritmi sa što većom pouzdanošću ispravno klasificiraju nepoželjne automobile, klasi *unacc* daje se veći značaj, stoga se kroz ovaj rad rezultati metrika računaju na temelju težinskog prosjeka.

AUC (eng. Area Under the Curve) označava područje ispod krivulje, tako da je pri izračunu ROC AUC rezultata prvo potrebno definirati ROC (eng. Receiver Operator Characteristic) krivulju, što je ponovo najlakše za objasniti putem binarne klasifikacije. Opoziv, kako je već ranije pojašnjen, predstavlja omjer točno predviđenih pozitivnih klasa u odnosu na stvaran broj pozitivnih klasa. S druge strane postoji i mjera tzv. specifičnosti (eng. specificity) koja je definirana kao omjer točno predviđenih negativnih klasa (TN) u odnosu na stvaran broj

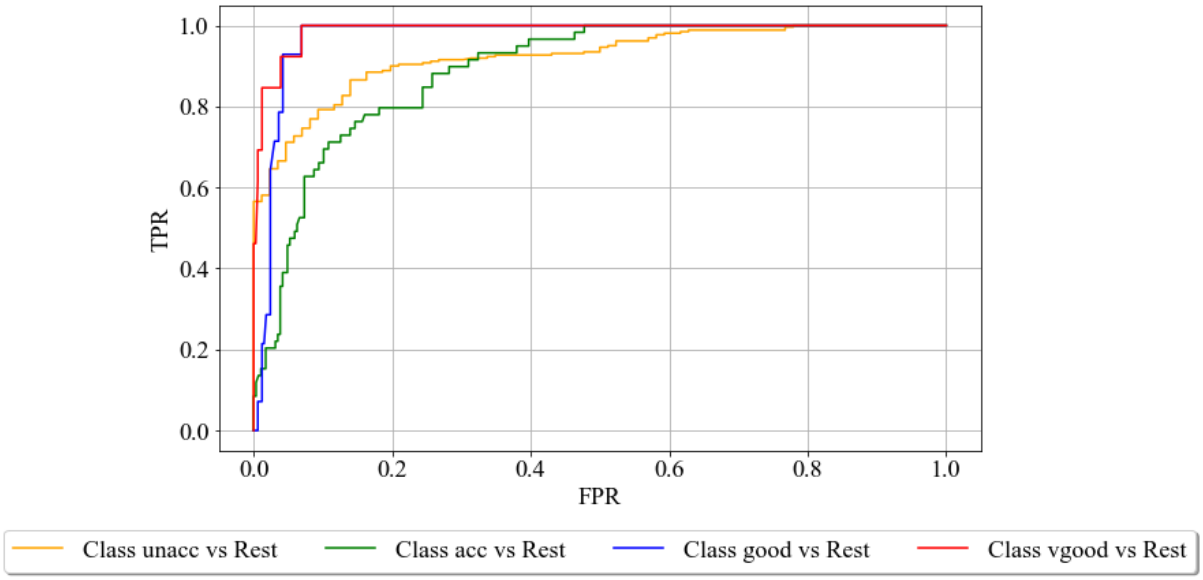
negativnih klasa (TN+FP). ROC krivulja prikazuje se kao omjer navedena dva parametra, pri čemu je os apscisa definirana kao komplementarna vrijednost specifičnosti, poznata još i kao lažno pozitivna stopa (eng. False Positive Rate (FPR)), dok se na osi ordinata nalazi opoziv, poznatiji kao stvarna pozitivna stopa (eng. True Positive Rate (TPR)). Primjeri ROC krivulje prikazani su slikom 4.3.



Slika 4.3. Primjer dviju ROC krivulja

Iz slike 4.3 se može uočiti kako je u rasponu visokog opoziva model s B krivuljom kvalitetniji, dok se u rasponu visoke specifičnosti model s A krivuljom ističe kao bolji odabir. Drugim riječima model B imalo bi puno više smisla upotrijebiti u slučajevima gdje se zahtjeva što manji broj lažno negativnih rezultata, poput testiranja ljudi na zarazne bolesti, dok bi model A bio korisniji primjeniti u situacijama gdje se zahtjeva što manji broj lažno pozitivnih rezultata, npr. izbjegavanje lažnih alarmnih stanja. No, značajnije svojstvo od samog oblika krivulje jest površina koja se nalazi ispod nje. Površina ispod krivulje (AUC) može poprimiti bilo koju vrijednost između 0 i 1. Pritom vrijednost 0 označava klasifikatora koji nije u stanju ispravno klasificirati niti jednu instancu. Ukoliko se između točaka (0,0) i (1,1) nalazi ravna linija, tada je AUC vrijednost jednaka 0.5 te je riječ o tzv. slučajnom klasifikatoru čija sposobnost predviđanja odgovara nasumičnom pogađanju. Takav klasifikator se obično koristi kao prag koji odvaja beskorisne modele od onih korisnih čija je ocjena bolja što se više krivulja približava vrijednosti (0,1).

Iz prethodnog opisa može se ustanoviti kako se ROC AUC rezultat općenito koristi za binarnu klasifikaciju i pokazuje koliko je model dobar u razlikovanju pozitivnih i negativnih ciljnih klasa. U slučaju pak problema višeklasne klasifikacije može se koristiti tehnika usporedbe svake klase s grupiranim preostalim klasama, poznata i kao OVR (eng. One Vs Rest) metoda. Takvom metodom generirane su ROC krivulje korištenjem istog modela kao za dobivanje matrice sa slike 4.2. te su prikazane slikom 4.4.



Slika 4.4. ROC krivulje generirane OVR metodom za "car evaluation" skup podataka

Krivulje se slike 4.4 se jasno daju do znanja kako algoritam najbolje razlikuje klase *good* i *vgood* u slučaju da se klase pojedinačno uspoređuju s preostalim klasama, stoga će za njihov slučaj AUC vrijednost biti veća nego kod klasa *acc* i *unacc*. No, kao i kod prethodno spomenutih metrika, za potrebe ovog rada AUC rezultat računat će se na temelju težinskog prosjeka tokom testiranja algoritama.

4.3. Generalizacija modela strojnog učenja

Iako se nakon podjele skupa podataka na podatke za treniranje i testiranje očekuje da će naučeni modeli sa 100%-tnom točnošću predviđati podatke na kojima su istrenirani, u praksi se takav ishod smatra vrlo problematičnim te su takvi modeli potpuno beskorisni. Razlog tomu jest što se u tom slučaju algoritam pretjerano prilagođava podacima za učenje. Drugim riječima, skup podataka za treniranje uči napamet, što rezultira savršenim rezultatima na

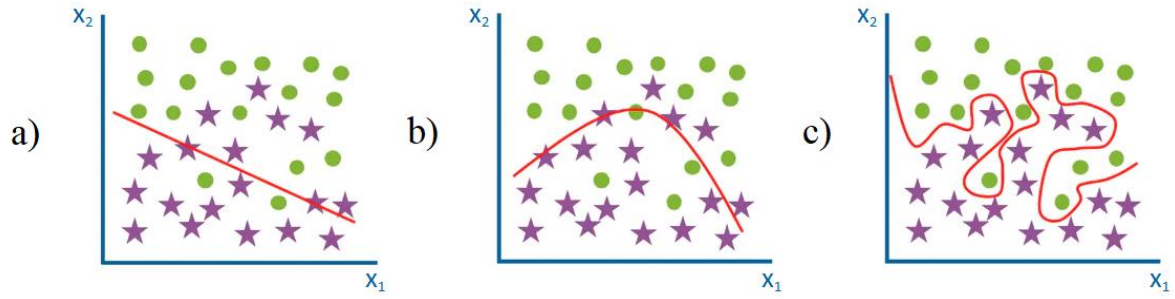
skupu za treniranje i slučajnim pogađanjem na skupu za testiranje. U području strojnog učenja takva se pojava naziva prenaučenost (eng. overfitting). Overfitting se može pojavljivati iz brojnih razloga, a jedan on njih je kada model uči detalje i šum koji se pojavljuju u podacima za treniranje, što kasnije negativno utječe na performanse modela kod predviđanja ishoda novih podataka [9].

Prisutnost tzv. šumova dovodi do komplikacija prilikom učenja. Naime, u situaciji većeg broja šumova postoje velike šanse da se ti šumovi nauče i kasnije služe kao temelj predviđanja [10]. U problemima klasifikacije postoje dvije vrste prisutnih šumova; šum klase i šum značajki. Šumu klase pripadaju pogrešno označene instance ili pak kontradiktorne instance koje se pojavljuju više puta u skupu podataka i označene su različitim oznakama klase, dok se za šum značajki smatra pogrešnost ili pak nepotpunost vrijednosti značajki [11]. Povećanjem količine podataka postoje veće šanse da će algoritam bolje funkcionirati, odnosno trebao bi moći s većom točnošću razlikovati reprezentativne podatke od šumova. Nažalost zbog mogućnosti da se šumovi pojavljuju i u podacima koji se dodatno unose postoji šansa da se rezultati samo dodatno pogoršaju, stoga je poželjno da takvi podaci budu unaprijed provjereni.

Problem prekomjernog učenja može biti prouzročen i povećanim brojem značajki zbog čega model postaje previše složen jer nastoji uzeti u obzir sve značajke, iako neke od njih nemaju prevelik utjecaj na izlaznu vrijednost [10]. Smanjenje njihovog broja rijetko kad može biti zadovoljavajuće rješenje s obzirom da nije uvijek poznato koje su značajke beskorisne. U slučaju linearnih modela kao rješenje se nudi metoda regularizacije, koja će biti opisana u narednom poglavlju.

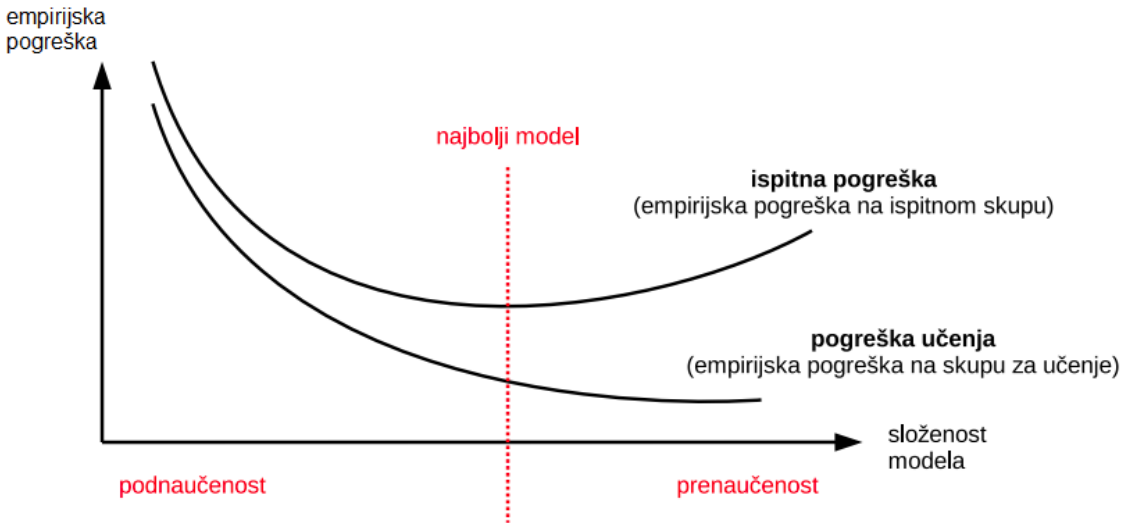
Suprotno od overfittinga, problem koji se također može pojaviti jest mogućnost da model ne uspjeva naučiti dovoljno iz podataka o obuci, što rezultira pojavom nedovoljnog opremanja (eng. underfitting). Uz to što će imati lošiju izvedbu na skupu za obuku, takav model neće biti u stanju dobro generalizirati nove podatke. Općenito, problemi prekomjernog i nedovoljnog opremanja u strojnom učenju opisuju se kroz pojmove pristranosti (eng. bias) i varijance (eng. variance). Pristranost se veže uz nesposobnost modela da točno procjeni pravi odnos između podatkovnih točaka u fazi učenja. Visoka pristranost obično se pojavljuje kod jednostavnih modela, kao što linearni modeli, koji nisu u stanju dovoljno dobro klasificirati instance preko kojih su se trenirali, a samim time ni nove teste instance. U slučaju pak vrlo niske pristranosti model postaje sklon učenju detalja, što rezultira overfittingom. Za takav model kaže se da je

nestabilan, odnosno da ima visoku varijancu. Visoka varijanca obično je prisutna kod složenih modela koji će, ovisno o ulaznom skupu podataka, nekad proizvesti dobra, a nekad vrlo loša rješenja. Utjecaj pristranosti i varijance na granicu odluke (eng. decision boundary), tj. na sposobnost razlikovanja klasa podatkovnih točaka prikazan je slikom 4.5.



Slika 4.5. Primjer utjecaja a) nedovoljno opremljenog, b) optimalnog i c) prekomjerno opremljenog modela na granicu odluke [12]

Sa slike 4.5 primjetno je kako se složenost modela povećava od lijevog grafa prema desnom. Visoka pristranost odgovara najjednostavnijem modelu (slika 4.5.a) koji nije u stanju dovoljno dobro razlikovati podatkovne točke dviju klasa, dok je s druge strane najsloženiji model (slika 4.5.c), koji ima najvišu varijancu, dosegao prekomjerno opremanje zbog čega će doći do velike fluktuacije uslijed klasificiranja novih instanci. Stoga se za najbolje rješenje odabire model u sredini (slika 4.5.b) koji će s velikom vjerojatnošću ostvariti dobre rezultate prilikom testiranja. Iz ovih zapažanja proizlazi jedan od najbitnijih grafova u području strojnog učenja, predočen slikom 4.6.



Slika 4.6. Graf utjecaja složenosti modela na empirijsku pogrešku [13]

Kao što je ranije navedeno, jednostavni modeli (visoka pristranost i niska varijanca) rezultirati će većom pogreškom učenja, a samim time i velikom ispitnom pogreškom, dok će previše složeni modeli (visoka varijanca i niska pristranost) imati malu pogrešku na skupu za učenje, ali znatno veću pogrešku na ispitnom skupu. Stoga, budući da su pristranost i varijanca obrnuto proporcionalni, glavni cilj prilikom odabira modela strojnog učenja jest naći kompromis između pristranosti i varijance, odnosno model koji će rezultirati najmanjom prosječnom pogreškom, kao što je prikazano na slici 4.6.

Kao rješenja pojedinih modela koja su implementirana u ovom radu promatraju se srednje vrijednosti rezultata dobivenih na skupu za treniranje i testiranje, kao i srednje vrijednosti standardnih devijacija, ne bi li se prepoznala eventualna pojava nedovoljnog ili prekomjernog opremanja.

5. ALGORITMI STROJNOG UČENJA

Postupak učenja i testiranja primjenit će se na neke od poznatijih klasifikasijskih algoritama strojnog učenja te će se na temelju dobivenih rezultata dobiti predodžba njihove kvalitete. Pritom će se za svaki model realizirati petlja u kojem će algoritam nasumično birati vrijednosti hiperparametara kako bi ostvario što bolji rezultat svake metrike. Za dobivanje konačnih rezultata pojedinog klasifikatora uzeti će se u obzir kombinacija hiperparametara koja je na izlazu dala navišu srednju vrijednost ROC AUC rezultata, nakon čega će se rezultati usporediti s rezultatima dobivenim uporabom defaultnih, tj. zadanih vrijednosti hiperparametara.

5.1. Klasifikator logističke regresije

Logistička regresija je algoritam strojnog učenja koji se koristi za probleme klasifikacije i temelji se na konceptu vjerojatnosti. Zbog svoje jednostavne implementacije i brzine klasificiranja novih podataka jedan je od najčešće korištenih linearnih modela. Iz ulaza izvlači realne vrijednosti značajki, svaku množi težinom, zbraja ih i prosljeđuje zbroj kroz sigmoidalnu funkciju za generiranje vjerojatnosti [14]. Koristi se kada je zavisna (izlazna) varijabla kategorička.

Cilj je istrenirati algoritam koji će moći donijeti binarnu odluku o klasi novog uzorka x, predstavljenog kao vektor značajki $[x_1, x_2,..., x_n]$. Izlaz y može poprimiti vrijednost 0 ili 1, stoga se promatra vjerojatnost $P(y = 1|x)$ da ulazni podatak pripada promatranoj klasi, odnosno vjerojatnost $P(y = 0|x)$ da ulazni podatak ne pripada promatranoj klasi. Podaci sa kojima se vrši treniranje ne mogu se mijenjati, međutim moguće je utjecati na parametre w (eng. weight) i b (eng. bias) pomoću kojih se zapravo predstavljaju težine i pomaci (odstupanja) [9]. Težina w_i, pridružena ulaznoj značajki x_i, realan je broj koji predstavlja koliko je značajka ulaza važna za odluku o klasifikaciji. Odluka o testnoj instanci donosi se na osnovu vrijednosti varijable z koja predstavlja zbroj ponderiranih značajki uz dodatak pomaka, stoga formula za izračun glasi [14]:

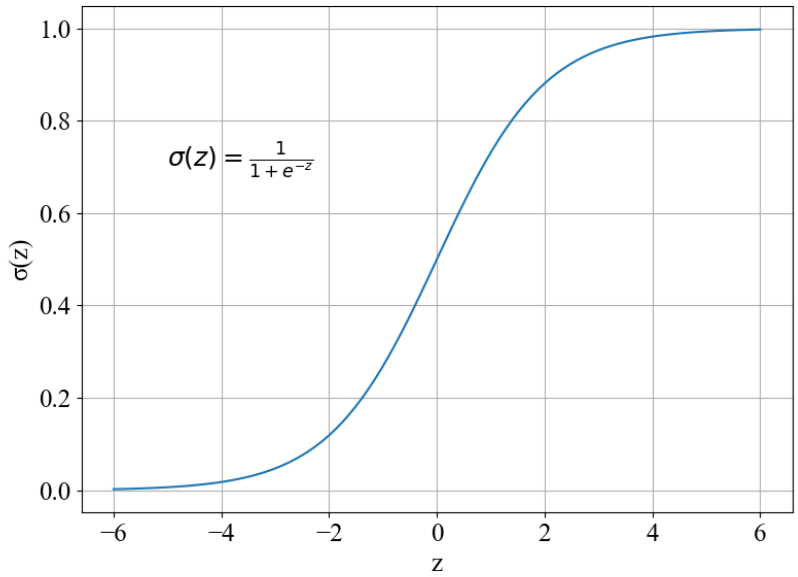
$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b\,, \tag{5.1}$$

ili jednostavnije:

$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b\ . \tag{5.2}$$

Primjenom prethodne formule z postaje realan broj te može poprimiti bilo koju vrijednost iz intervala od od $-\infty$ do ∞. Stoga, da bi se dobila vjerojatnost, tj. vrijednost unutar intervala od 0 do 1, z se provodi kroz sigmoidalnu funkciju prikazanu slikom 5.1. i definiranu formulom:

$$\sigma(z) = \frac{1}{1 + e^{-z}}\,. \tag{5.3}$$



Slika 5.1. Prikaz sigmoidalne funkcije

Sigmoidalna funkcija, koja se još naziva i logističkom funkcijom, uzima realan broj i preslikava ga u raspon [0,1]. Zahvaljujući svom obliku funkcija će testnu instancu s većom vjerojatnošću točno klasificirati što je više vrijednost z udaljenija od nule. Vjerojatnost da ulazni podatak pripada promatranoj klasi računa se izrazom:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}}\,, \tag{5.4}$$

gdje je:

σ(z) – sigmoidalna funkcija,

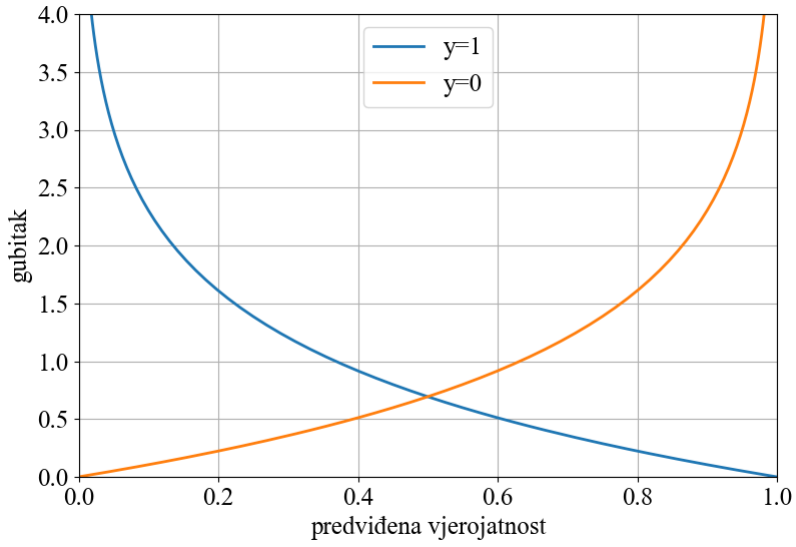
$\boldsymbol{x}$ – vektor ulaznog podatka,

$\boldsymbol{w}$ – vektor težine i

b – pomak.

Odluka o tome kojoj klasi se dodjeljuje testna instanca x donosi se uz pomoć praga koji je definiran vrijednošću 0.5. Ukoliko je vjerojatnost $\hat{y}$ veća od 0.5 instanca se dodjeluje klasi 1, u suprotnom dodjeljuje se klasi 0.

Prilikom analiziranja logističke regresije obično je od koristi znati koliko se predviđene vrijednosti klasifikatora razlikuju od pravih vrijednosti (0 ili 1). Njihovo odstupanje opisuje se korištenjem funkcije pogreške (eng. cost function). Funkcija pogreške uzima izlaze predviđene modelom i stvarne izlaze te izračunava koliko je model bio u krivu prilikom predviđanja. Definirana je kao prosjek tzv. funkcija gubitka na skupu uzoraka. Funkcija gubitka (eng. loss function) pak predstavlja pogrešku prilikom promatranja jedne podatkovne točke. Pritom postoje različite metode za izračun odstupanja, no za probleme klasifikacije najčešće se koristi unakrsna entropija (eng. Cross Entropy). Gubitak unakrsne entropije definiran je kao logaritam vjerojatnosti izlaza koji odgovara ispravnoj klasi [14]. Efikasnost proizlazi iz logaritamskog oblika grafova, čime se nastoji jače "kažnjavati" veća odstupanja od pravih vrijednosti. Prikaz grafova unakrsne entropije predočen je slikom 5.2.



Slika 5.2. Prikaz grafova unakrsne entropije

Kao što se može primjetiti sa slike 5.2, kod izračuna gubitka promatraju se dva grafa, jedan u slučaju da je prava izlazna vrijednos 1 (plavi graf), a drugi u slučaju da je prava izlazna

vrijednost 0 (narančasti graf). Ukoliko je prava vrijednost 1 gubitak je manji što je predviđena vjerojatnost bliža vrijednosti 1, odnosno veći ukoliko je predviđena vjerojatnost bliža 0. Ista metodologija vrijedi i ako je prava vrijednost 0, gubitak je manji što je predviđena vjerojatnost bliža pravoj vrijednosti, i obrnuto. Na temelju ovih svojstava definirana je jedinstvena formula funkcije gubitka:

$$L_{CE}(\hat{y}, y) = -\ln p(y|x) = -[y \ln(\hat{y}) + (1-y)\ln(1-\hat{y})]\,, \tag{5.5}$$

koja nakon uvođenja izraza $\hat{y} = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ poprima oblik:

$$L_{CE}(\hat{y}, y) = -\big[y \ln\big(\sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)\big) + (1-y)\ln\big(1-\sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)\big)\big]\,, \tag{5.6}$$

gdje je:

L_{CE} – funkcija gubitka unakrsne entropije,

y – točna vrijednost,

$\hat{y}$ – predviđena vjerojatnost,

$\boldsymbol{x}$ – vektor ulaznog podatka,

$\boldsymbol{w}$ – vektor težine i

b – pomak.

Dva člana unutar zagrade odnose se na dva moguća slučaja; $y = 0$ i $y = 1$. Kada je $y = 0$, prvi član nestaje i ostaje samo drugi član. Isto tako, kada je $y = 1$, drugi član nestaje i ostaje samo prvi član. Konačno, funkcija pogreške, tj. pogreška unakrsne entropije glasi:

$$L(\boldsymbol{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i)]\,, \tag{5.7}$$

gdje je:

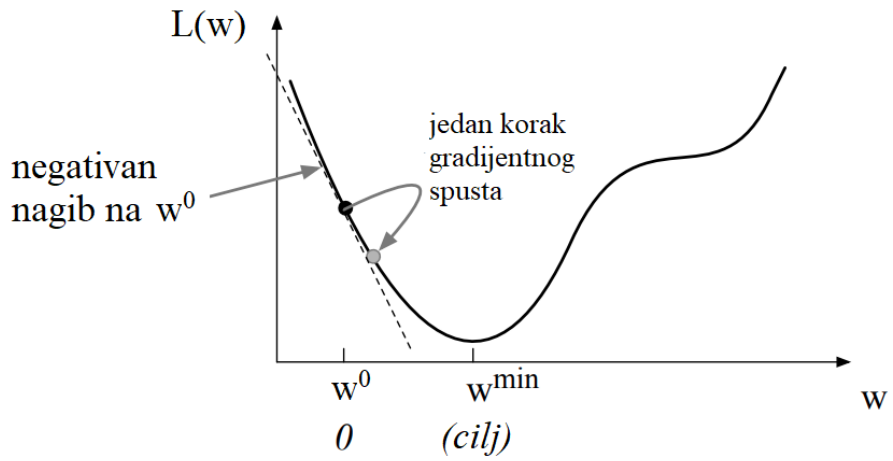
y – točna vrijednost,

$\hat{y}$ – predviđena vjerojatnost i

N – broj uzoraka.

Nakon definiranja funkcije pogreške cilj je minimizirati gubitke na način da se optimiziraju parametri modela. Optimizacija parametara postići će se pomoću metode gradijentnog spusta

(eng. gradient descent method). Gradijentni spust je iterativni optimizacijski algoritam prvog reda koji pronalazi minimum funkcije određivanjem u kojem smjeru, u prostoru parametara, nagib funkcije raste najstrmije, nakon čega se kreće suprotno od smjera rasta [14]. Nasumičnim odabirom težine w na nekoj vrijednosti w^0 algoritmom gradijentnog spusta dobiti će se informacija o tome treba li se u sljedećoj iteraciji pomaknuti ulijevo, tj. umanjiti w^1 u odnosu na w^0, ili udesno, odnosno povećati w^1 u odnosu na w^0, da bi se postigao minimum. Primjer izvođenja algoritma gradijentnog spusta prilikom ovisnosti funkcije pogreške o samo jednom parametru (jednoj težini) prikazan je slikom 5.3.



Slika 5.3. Primjer izvođenja algoritma gradijentnog spusta kod ovisnosti o jednom parametru [14]

Na slici 5.3 vidi se proces u kojem se, zahvaljujući izračunu gradijenta funkcije pogreške u nasumično odabranoj početnoj točki w^0, dobiva informacija o negativnom nagibu zbog čega se u slijedećem koraku povećava vrijednost *w*. Iznos za koji će se trebati umanjiti/uvećati vrijednost *w* u gradijentnom spuštanju određen je vrijednošću nagiba, tj. gradijenta, pomnoženog sa stopom učenja η. Stoga jednadžba izvršavanja metode gradijentnog spusta glasi:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla L(\boldsymbol{w}_t), \tag{5.8}$$

gdje je:

$\boldsymbol{w}_t$ – trenutna vrijednost parametra težine,

$\boldsymbol{w}_{t+1}$ – slijedeća vrijednost parametra težine,

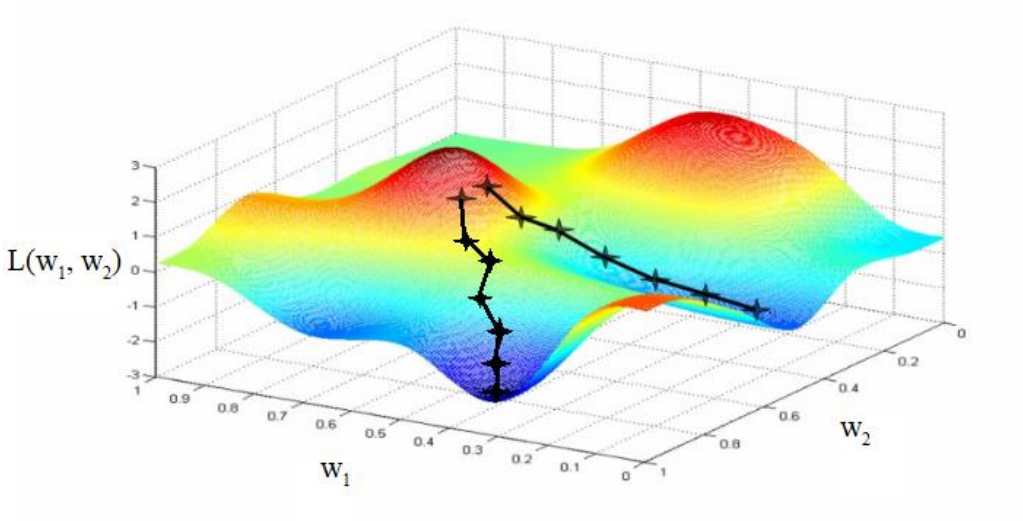
η – stopa učenja i

$L(w)$ –funkcija pogreške.

Stopa učenja određuje veličinu koraka kod spuštanja prema minimumu. Odabir odgovarajuće stope učenja je od velike važnosti budući da niska stopa učenja za posljedicu ima povećanje vremena potrebnog za postizanje minimuma, dok visoka stopa učenja lako može uzrokovati prekoračenje i udaljavanje od minimuma, što rezultira visokim gubicima. Budući da u primjeru sa prethodne slike funkcija ovisi o jednom parametru svaki idući parametar (težina) računa se jednostavnom formulom:

$$w_{t+1} = w_t - \eta \frac{\partial L(w)}{\partial w_t}. \tag{5.9}$$

Ipak, u većini slučajeva raspolaže se s funkcijom koja ovisi o više parametara. U tom slučaju potrebno je znati kretati se u N-dimenzionalnom prostoru. Korištenjem gradijenta moguće je izraziti komponente smjera najoštrijeg nagiba duž svake od tih N dimenzija [14]. Slika 5.4 prikazuje pretragu minimuma funkcije pogreške u ovisnosti o dva parametra težine.



Slika 5.4. Primjer izvođenja algoritma gradijentnog spusta kod ovisnosti o dva parametra [15]

Sa slike 5.4 primjećuje se kako odabir početne točke zna biti presudan jer postoji mogućnost da algoritam prilikom traženja optimuma zaglavi u lokalnom minimumu. Gradijent funkcije s više varijabli predstavlja vektor u kojem svaka komponenta izražava parcijalnu derivaciju funkcije u odnosu na jednu od varijabli [14]. Stoga za gradijent funkcije pogreške u slučaju većeg broja parametara vrijedi formula:

$$\nabla L(\boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}} L(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(\boldsymbol{w}) \\ \frac{\partial}{\partial w_2} L(\boldsymbol{w}) \\ \vdots \\ \frac{\partial}{\partial w_n} L(\boldsymbol{w}) \end{bmatrix}. \tag{5.10}$$

Kao što je već spomenuto u prethodnom poglavlju, algoritmi mogu biti skloni prekomjernom opremanju uslijed povećanog broja značajki. U slučaju linearnog modela, kao što je logistička regresija, taj se problem može reducirati primjenom metode regularizacije. Regularizacija nastoji minimizirati težine značajki koje imaju mali utjecaj na konačnu klasifikaciju [10]. Da bi se to postiglo funkciji pogreške dodaje se "kazna", tzv. regularizator. U dvije najčešće korištene regularizacije ubraja se L1 regularizacija, definirana formulom:

$$L_{L1}(w) = L(w) + \frac{\lambda}{2} \sum_{j=1}^{n} |w_j|, \tag{5.11}$$

i L2 regularizacija, definirana formulom:

$$L_{L2}(w) = L(w) + \frac{\lambda}{2} \sum_{j=1}^{n} w_j^2, \tag{5.12}$$

gdje je:

L_{L1} – regularizacija funkcije pogreške L1 normom,

L_{L2} – regularizacija funkcije pogreške L2 normom,

L – funkcija pogreške,

w – parametar težine,

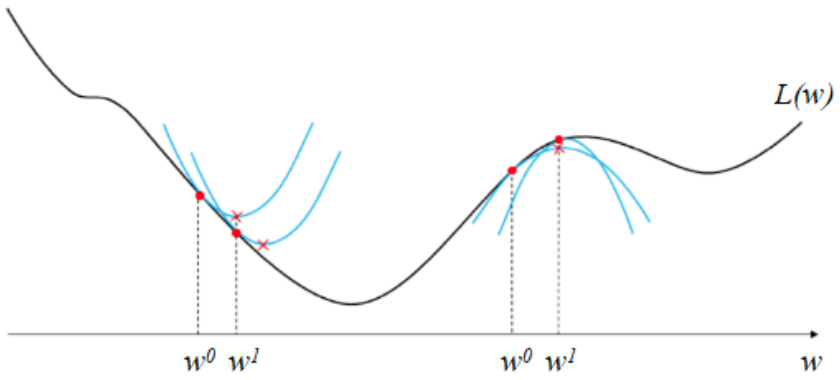
λ – parametar regularizacije i

n – broj uzoraka.

Iz jednadžbe 5.11 primjetno je kako član regularizacije odgovara funkciji apsolutne vrijednosti, što bi značilo da će se prilikom iterativnog postupka gradijentnog spusta gradijent člana regularizacije nastojati pomaknuti vrijednost težine prema 0 za konstantan korak čiji iznos ovisi o vrijednosti parametra λ. Nasuprot tomu, član regularizacije u jednadžbi 5.12 odgovara kvadratnoj funkciji, zbog čega će se vrijednost težine također nastojati pomaknuti

prema 0, ali za različite iznose koraka, budući da je gradijent kvadratne funkcije linearna funkcija. Drugim riječima, obje vrste regularizacije pokušavaju umanjiti vrijednosti težina, no glavna razlika je u tome što u slučaju L2 norme težine se nastoje pomicati prema 0, ali tako da se u svakoj idućoj iteraciji veličina koraka smanjuje, zbog čega težine nikada neće biti u potpunosti biti jednake 0. S druge strane to nije slučaj kod L1 regularizacije koja primjenjuje konstantan korak, zbog čega će neke od težina u jednom trenutku svesti na iznos 0, što će rezultirati eliminacijom određenih značajki. Odgovarajuća vrsta regularizacije stoga se odabire ovisno o tome želi li se ostvariti rijetkost u težinama i na taj način smanjiti složenost modela ili je pak potrebno zadržati sve parametre, ali svejedno nastojati umanjiti mogućnost pojave overfittinga.

Preko parametra regularizacije λ određuje se snaga regularizacije. Unutar programskog paketa Python taj je parametar naveden u inverznom obliku, $C = 1/\lambda$. Manja vrijednost parametra C odgovarati će jačoj regulaciji, odnosno smanjenju težinskih koeficijenata. Osim ugađanja hiperparametara kazne (L1 i L2) i vrijednosti C, klasifikator logističke regresije težiti će ka optimalnim rješenjima uz podešavanje vrijednosti maksimalnog broja iteracija dostupnih za konvergenciju i tolerancije za kriterij zaustavljanja. Također, omogućen je i odabir rješavača (eng. solver) optimizacijskih problema koji predstavlja različite alternative gradijentnog spuštanja, pri čemu je posebno zanimljiva Newtonova metoda koja koristi kvadratnu funkciju za aproksimaciju funkcije pogreške. Primjer Newtonove metode optimizacije prikazan je slikom 5.5.



Slika 5.5. Primjer izvođenja Newtonove metode optimizacije [16]

Kao što se vidi na slici 5.5, nova vrijednost težine dobiva se na osnovu korijena kvadratne funkcije iz prethodne iteracije. Metoda je učinkovitija od gradijentnog spuštanja u smislu da

zahtijeva mnogo manje koraka za konvergenciju. Međutim, uz to što je što je računalno zahtjevnija, u slučaju nekonveksnih funkcija pogreške kvadratne aproksimacije mogu biti konkavne te imaju tendenciju zaglaviti u lokalni maksimum.

```
def LogRegParRandomSearch():
    parameters = []
    #Choosing penalty
    Penalty = random.choice(['l1','l2', 'none'])
    #Choosing Tolerance
    Tol = random.uniform(1e-10, 1e-4)
    #Choosing inverse of regularization strength
    C = random.uniform(0, 1)
    #Choosing solver
    while True:
        Solver = random.choice(['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'])
        if (Solver == 'newton-cg') and (Penalty == 'l2' or Penalty == 'none'):
            break
        elif (Solver == 'lbfgs') and (Penalty == 'l2' or Penalty == 'none'):
            break
        elif (Solver == 'liblinear') and (Penalty == 'l1' or Penalty == 'l2'):
            break
        elif (Solver == 'sag') and (Penalty == 'l2' or Penalty == 'none'):
            break
        elif (Solver == 'saga'):
            break
        else:
            continue
    #Choosing maximum number of iterations
    MaxIter = random.randint(100,1000)
    ##############################################
    #Appending Randomly Selected Data into Parameters list
    ##############################################
    parameters.append(Penalty)
    parameters.append(Tol)
    parameters.append(C)
    parameters.append(Solver)
    parameters.append(MaxIter)

    return parameters
```

5.2. SGD klasifikator

Općenito govoreći SGD klasifikator ne predstavlja određenu vrstu modela već je riječ o linearnom klasifikatoru koji je optimiziran pomoću stohastičkog gradijentnog spusta (eng. Stochastic Gradient Descent (SGD)). U prethodno opisanom modelu dobio se uvid u minimizaciju funkcije pogreške korištenjem gradijentnog spusta u kojem su se parametri iterativnom metodom nastojali optimizirati. U svakom koraku iteracije parametar se umanjivao za uzmožak stope učenja i gradijenta funkcije pogreške pritom koristeći cjelokupan skup uzoraka za obuku. Na velikom skupu podataka to postaje računski vrlo

zahtjevno, što rezultira sporom konvergencijom. U tom se slučaju kao rješenje nudi SGD optimizacija. SGD je stohastička metoda optimizacije koja koristi stohastičku aproksimaciju gradijentnog spusta [9]. Stohastički znači da u svakom koraku algoritam napravi gradijent gubitka slučajno odabrane podatkovne točke (instance) za obuku, što ga čini mnogo bržim u odnosu na gradijentni spust koji u svakom koraku napravi gradijent gubitaka za sve primjere iz skupa za učenje, nakon čega se usmjerava optimalnom putanjom prema minimumu funkcije pogreške. SGD algoritam za ažuriranje težine stoga je definiran formulom:

$$w_{t+1} = w_t - \eta \frac{\partial L_i(w)}{\partial w_t} \tag{5.13}$$

gdje je:

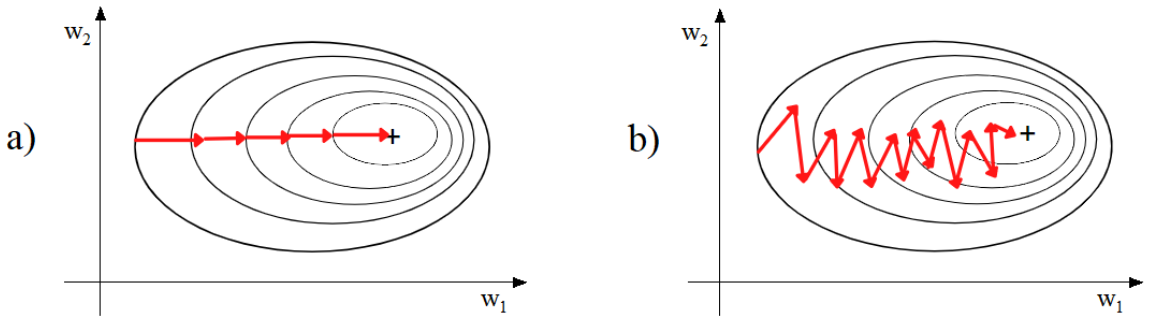
w_t – trenutna vrijednost parametra težine,

w_{t+1} – slijedeća vrijednost parametra težine,

η – stopa učenja i

$L_i(w)$ –funkcija pogreške i-te podatkovne točke.

Računanje preko cijelog skupa za obuku u svakoj iteraciji rezultira sporim koračanjem prema globalnom optimumu, što bi značilo da će se u istom periodu SGD-om moći ostvariti veći broj koraka. Na slici 5.6. prikazani su primjeri kretanja prema globalnom optimumu za navedene algoritme optimizacije.



Slika 5.6. Primjer kretanja prema globalnom optimumu a) gradijentnim spustom i b) SGD-om [17]

Na slici 5.6 funkcija pogreške predočena je pomoću konturnih linija, gdje je globalni optimum označen simbolom +. Iako ostvaruje bržu kretnju potrebno je naglasiti kako su ažuriranja stohastičkog gradijentnog spusta samo estimacije stvarnog gradijenta [9], stoga

putanja koračanja prema minimumu funkcije nije optimalna. Premda se čini kako je osciliranje prilikom traženja globalnog optimuma nedostatak SGD algoritma, dapače u nekim slučajevima to može biti od koristi. U situaciji kada funkcija pogreške nije konveksna, odnosno ima prisutne lokalne minimume, kao na slici 5.4, u tom slučaju stohastičko krivudanje pri spustu može pomoći da algoritam ne zaglavi u lokalnom minimumu [18].

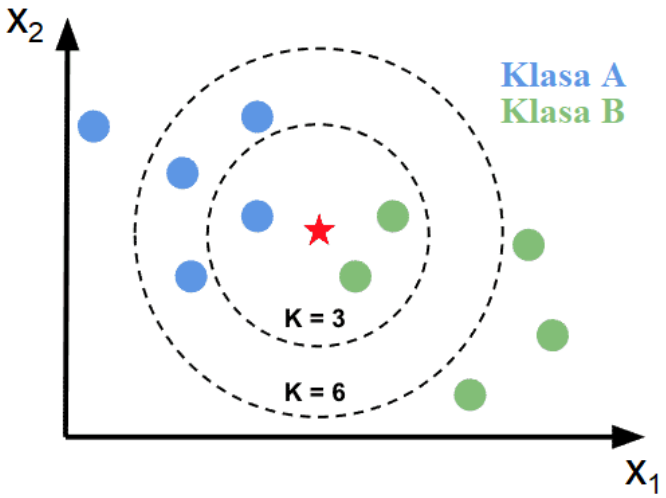
Kao što je objašnjeno na početku, SGD spada u metodu optimizacije kojom se minimizira funkcija pogreške. Prilikom implementacije modela u programskom jeziku Python omogućen je odabir različitih vrsta funkcija gubitka. Uz unakrsnu entropiju dopustiti će se odabir funkcija čijom primjenom je također moguće na izlazu dobiti vrijednosti predviđenih vjerojatnosti, koji su kasnije potrebni za izračun ROC AUC rezultata. Hiperparametri koji se ugađaju isti su kao kod klasifikatora logističke regresije. Uz njih se još nastoji podesiti i hiperparametar koji definira maksimalan broj iteracija dopuštenih za poboljšanje pogreške. Ukoliko u zadanom broju iteracija razlika u pogrešci ne bude manja od iznosa tolerancije podešavanje parametara se zaustavlja.

```
def SGDParRandomSearch():
    parameters = []
    #Choosing loss function
    Loss = random.choice(['log_loss', 'modified_huber'])
    #Choosing penalty
    Penalty = random.choice(['l2', 'l1'])
    #Choosing constant that multiplies the regularization term
    Alpha = random.uniform(1e-4, 1e-1)
    #Choosing maximum number of iterations
    MaxIter = random.randint(1000,10000)
    #Choosing Tolerance
    Tol = random.uniform(1e-9, 1e-3)
    #Maximum number of iterations without change
    while True:
        nIter = random.randint(5,10000)
        if nIter < MaxIter:
            print("nIter smaller than MaxIter")
            break
        else:
            continue
    ################################################
    #Appending Randomly Selected Data into Parameters list
    ################################################
    parameters.append(Loss)
    parameters.append(Penalty)
    parameters.append(Alpha)
    parameters.append(MaxIter)
    parameters.append(Tol)
    parameters.append(nIter)

    return parameters
```

5.3. Klasifikator k-najbližih susjeda i radijusa susjeda

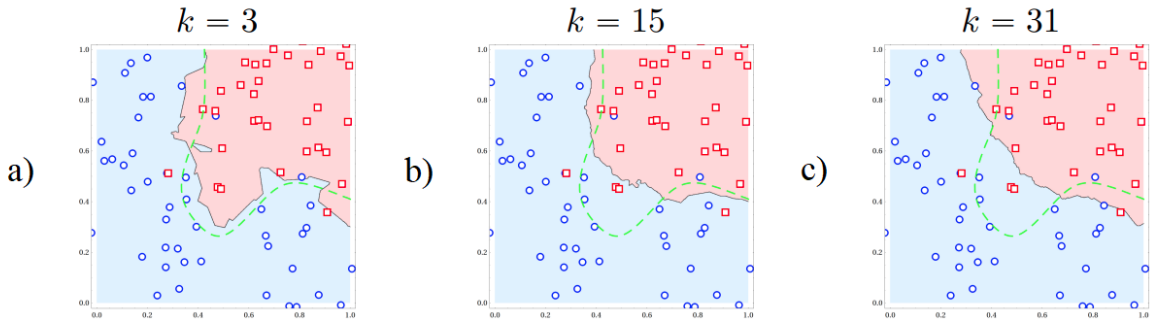
Algoritam k-najbližih susjeda (eng. K-Nearest Neighbors (KNN)) predstavlja vrlo jednostavan algoritam zbog čega je jedan od nakorištenijih modela strojnog učenja. Spada u tzv. lijene algoritme učenja (eng. Lazy learning algorithms). Za razliku algoritama željnog učenja (eng. Eager learning algorithms), u koje spadaju logistička regresija i algoritami koji će se analizirati u narednim poglavljima, koji u fazi učenja nastoje optimizirati parametare koji opisuju odnos između ulaznih i izlaznih varijabli, lijeni algoritmi ne sadrže eksplicitno fazu učenja, tj. faza učenja služi im samo za pohranu skupa podataka za obuku. Drugim riječima, umjesto konstruiranja modela spremnog za klasificiranje testnih instanci, lijeni algoritmi jednostavno pohranjuju podatke za trening i čekaju dok ne dobiju testni skup podataka [19]. Također, KNN je algoritam neparametarske prirode, što znači da se predviđanje klasifikatora ne temelji na pretpostavkama između ulaznih i izlaznih podataka opisanih parametrima. Jedina pretpostavka KNN algoritma prilikom klasifikacije nove podatkovne točke jest da se instance sličnih vrijednosti nalaze u neposrednoj blizini. Nakon pohrane podatkovnih točki za obuku u n-dimenzionalan prostor uzoraka KNN klasifikator svaku novu instancu dodjeljuje određenoj klasi na temelju "glasa većine" (eng. majority vote), odnosno klasi koja prevladava u njenoj okolini. Primjer promatranja susjednih klasa prilikom unosa nove podatkovne točke prikazan je slikom 5.7.



Slika 5.7. Ilustracija rada KNN algoritma u 2D prostoru [20]

Najvažnija stvar prilikom implementacije KNN algoritma jest definiranje koeficijenta *k* koji označava broj promatranih susjeda instance koja se klasificira. Na slici 5.7 vidi se kako nova

točka promatra broj susjeda za dva različita slučaja, k = 3 i k = 6. Za slučaj k = 3 vrijedi da dvije od 3 susjedne točke pripadaju B klasi, stoga će instanca biti dodjeljena klasi B, dok će u slučaju k = 6 biti dodjeljena klasi A s obzirom da većina od 6 promatranih pripada klasi A. Iz primjera proizlazi jedan od temeljnih problema KNN algoritma, a to je odabir optimalne vrijednosti k koeficijenta. Mala vrijednost k može lako rezultirati krivom odlukom, budući da će u tom slučaju šumovi imati veći utjecaj na rezultat. Povećavanjem koeficijenta k vrijednosti granice između klasa postaju sve "glađe", kao što se može primjetiti na slici 5.8.



Slika 5.8. Primjer usporedbe rezultata klasifikacije za vrijednosti a) k = 3, b) k = 15 i c) k = 31 [21]

Iako se sa slike 5.8 vidi da povećanjem koeficijenta k nekolicina instanci postaje krivo klasificirana, to je zapravo korisno, budući da se time umanjuje vjerojatnost pretjeranog prilagođavanja, tj. overfittinga. Pritom treba imati na umu kako veća vrijednost zahtjeva i veću računalnu snagu. Općenito, način na koji se može barem minimalno umanjiti mogućnost pogreške jest da se odabere neparna vrijednost k koeficijenta. Neparnom vrijednošću koeficijenta k smanjuje se rizik od izjednačenja u odluci prilikom klasificiranja.

Udaljenosti između susjednih točaka mogu se izračunati na različite načine, no Euklidska udaljenost jedna je od najednostavnijih i najčešće korištenih metrika udaljenosti. Predstavlja mjeru stvarne, najkraće udaljenosti između dvaju vektora značajki, stoga se izračunava formulom:

$$D(\boldsymbol{x}, \boldsymbol{x}_i) = \sqrt{\sum_{j=1}^{n} \left(x_j - x_{i,j}\right)^2}\ , \tag{5.14}$$

gdje je:

$D(\boldsymbol{x},\boldsymbol{x}_i)$ – udaljenost između vektora,

$\boldsymbol{x}$ – vektor značajki i

n – broj dimenzija.

Prilikom klasificiranja nove podatkovne točke algoritam prolazi kroz cijeli skup primjera za učenje nebi li pronašao k-susjeda koja su mu najbliža. Ta metoda odgovara algoritmu grube sile (eng. brute force). Riječ je o osnovnom algoritmu koji uključuje izračunavanje udaljenosti prema svim podatkovnim točakama u prostoru podataka. Takav način pretrage čini model vrlo sporim, stoga se primjenjuje jedino u slučaju malih skupova podataka. Kao rješenje nudi se uporaba algoritama kojima se može ostvariti brža pretraga, među koje spadaju algoritmi k-d stabla i loptastog stabla, koji svoj proces temelje na podjeli podatkovnih točaka u određene regije.

Algoritam k-d stabla (eng. k-d tree) formira binarno stablo u kojem svaki čvor odgovara potpodručju s određenim brojem k-dimenzionalnih točaka. Tijekom konstrukcije stabla trenutni prostor značajki dijeli se na dva potprostora, pri čemu polovica točaka leži u svakom potprostoru. Ta se podjela temelji na trenutnoj dimenziji, nakon čega se algoritam prebacuje na sljedeću dimenziju i dobivene potprostore dijeli na dva dijela, pri čemu opet polovica točaka leži u svakom potprostoru. Proces se zaustavlja kada u svakom potprostoru preostane broj točaka koji je unaprijed definiran veličinom lista. Čvorovi stabla stoga imaju ulogu da postavljaju uvjete testnoj instanci usmjeravajući ju tako prema odgovarajućem potprostoru, tj. listu stabla u kojem bi se mogli nalaziti najbliži susjedi. Algoritam zatim najbliže susjede u odgovarajućem listu sprema kao trenutno najbolja rješenja. Mogućnost pronalaska bližih susjeda zatim ispituje na način da postepenim vraćanjem prema korijenu stabla pretražuje rješenja u potpodručjima čiji su rubovi na manjoj udaljenosti od trenutno spremljenih rješenja. Na taj način eliminira pretragu potpodručja čiji su rubovi udaljeniji od trenutno najboljih rješenja.

Glavni nedostatak k-d stabla je slabija učinkovitost prilikom većeg broja dimenzija. U tom slučaju korisnije je primjeniti algoritam loptastog stabla (eng. ball tree) koji formira binarno stablo u kojem svaki čvor predstavlja hipersferu, tj. "loptu" koja sadrži podskup točaka. Korijenski čvor stoga sadrži puni skup točaka iz skupa podataka, a svaki lisni čvor ima broj točaka koji je određen veličinom lista [21]. Svaki od čvorova definiran je centroidom, stoga

će testna instanca najbliže susjede tražiti u hipersferama čiji centroidi su na najmanjoj udaljenosti od njene pozicije.

Uz koeficijent *k* i algoritam za računanje najbližih susjeda prilikom izvođenja KNN algoritma unutar Python-a nastojat će se odabrati i odgovarajuća težinska funkcija, za koju se nude opcije *uniform* i *distance*. Vrijednost *uniform* označava ujednačenost težina točaka prilikom glasovanja, tj. svaki glas je jednako značajan prilikom klasificiranja, dok kod vrijednosti *distance* točkama (susjedima) koje su bliže testnoj instanci pridaje se veći značaj, odnosno imaju veći utjecaj na donošenje odluke.

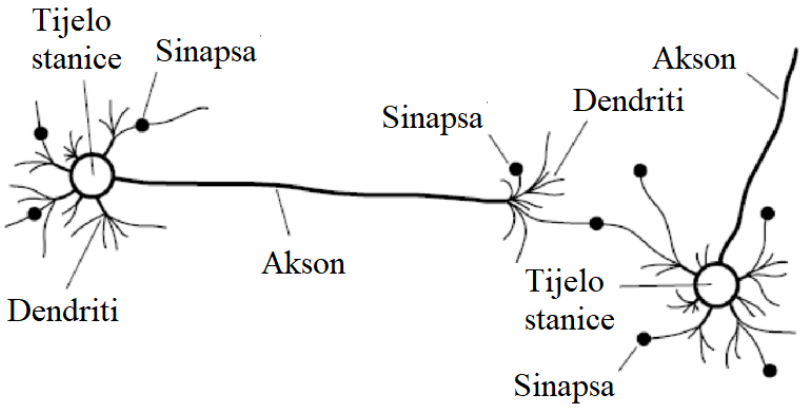
```
def KNNParRandomSearch():
    parameters = []
    #Choosing number of neighbors
    N_neighbors = random.randint(1,5)*2+1
    #Choosing weight function used in prediction
    Weights = random.choice(['uniform','distance'])
    #Choosing algorithm used to compute nearest neighbors
    Algorithm = random.choice(['auto', 'ball_tree', 'kd_tree', 'brute'])
    ##############################################
    #Appending Randomly Selected Data into Parameters list
    ##############################################
    parameters.append(N_neighbors)
    parameters.append(Weights)
    parameters.append(Algorithm)

    return parameters
```

Gotovo identičan proces djelovanja vrijedi kod još jednog modela, tzv. klasifikatora radijusa susjeda (eng. Radius Neighbors Classifier). Za razliku od KNN algoritma ovaj klasifikator provodi glasanje među susjedima unutar fiksnog radijusa. Iako je njegova primjena puno rjeđa od KNN-a, u situacijama kada podaci nisu ravnomjerno uzorkovani klasifikacija na temelju radijusa može biti bolji izbor. U tom slučaju spriječava se da znatno udaljeniji primjeri daju svoj doprinos prilikom glasanja. Ipak, za višedimenzionalne prostore parametara ova metoda postaje manje učinkovita zbog tzv. "prokletstva dimenzionalnosti" (eng. Curse of dimensionality). Izraz je to koji označava da će povećanje dimenzija, tj. značajki rezultirati eksponencijalnim rastom količine podataka potrebnih za generalizaciju modela. Višedimenzionalni prostori stvaraju velik problem prilikom uporabe mjere udaljenosti. Naime, kako dimenzionalnost prostora raste razmak između točaka postaje sve veći i točke imaju tendenciju da postanu jednako udaljene [22]. Drugim riječima, razlika između udaljenosti točaka praktički nestaje, stoga pretraga primjera uz pomoć fiksnog radijusa nije najbolji odabir.

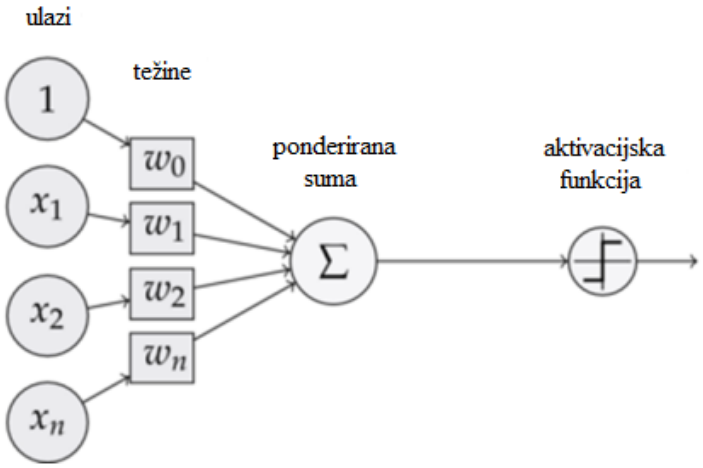
5.4. MLP klasifikator

MLP klasifikator je klasifikator čije se djelovanje temelji na radu višeslojnog perceptrona (eng. Multi-Layer Perceptron (MLP)). Višeslojni perceptron model je umjetne neuronske mreže koji preslikava ulazne skupove podataka u skup odgovarajućih izlaza. Inspirirane funkcioniranjem bioloških neurona, umjetne neuronske mreže (eng. Artificial Neural Networks (ANN)) pokušavaju oponašati rad ljudskog mozga koji predstavlja skup međusobno povezanih neurona. Dva povezana biološka neurona prikazana su slikom 5.9.



Slika 5.9. Biološka neuronska mreža [23]

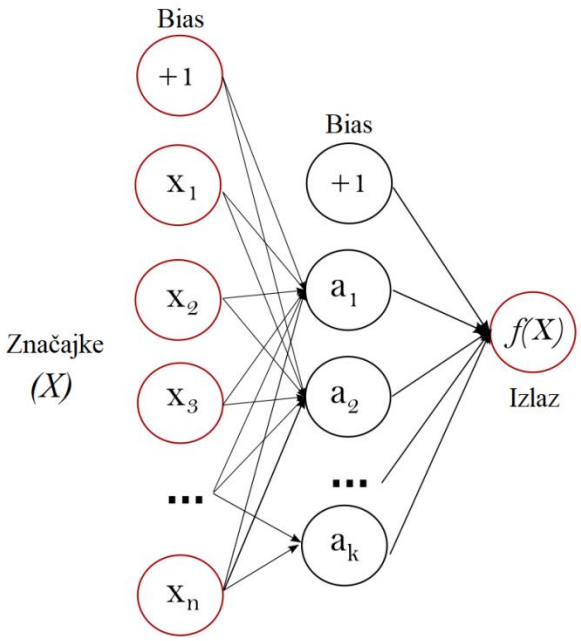
Biološki neuron stanica je koja prima informacije od drugih neurona putem dendrita, obrađuje ih, a zatim šalje impuls putem aksona i sinapsi drugim neuronima u mreži. Učenje se pritom odvija promjenom jačine sinaptičkih veza [24]. Analogno tomu, osmišljena je struktura umjetnog neurona kod kojeg su signali (ulazi) predstavljeni numeričkim vrijednostima, dok je jakost sinapse opisana težinskim faktorom w. Tijelo stanice pritom ima ulogu zbrajala, a akson je predstavljen prijenosnom (aktivacijskom) funkcijom. Primjer umjetnog neurona prikazan je slikom 5.10.



Slika 5.10. Prikaz umjetnog neurona [25]

Dolaskom signala do ulaza neurona signali se množe sa težinskim faktorima koji se dodjeljuju odgovarajućim ulazima. Ponderirani ulazni signali zatim se sumiraju, kao što je prikazano na slici 5.10. Također, sumi se dodaje i odstupanje (eng. bias) čime se ostvaruje bolja prilagodba modela na dane podatke. Kao i težinski faktori, i biasi se nakon svake iteracije u procesu učenja postepeno mijenjaju s ciljem da idući rezultat bude bliže željenom rezultatu [25]. Proces izračuna izlazne vrijednosti stoga je vrlo sličan kao kod logističke regresije. Razlika u odnosu na logističku regresiju je u tome što aktivacijska funkcija, tj. funkcija koja preslikava rezultat sumacije u izlazni signal (predviđenu vjerojatnost) ne mora nužno biti logistička, već se mogu koristiti različite funkcije, poput ReLU ili tangens hiperbolne funkcije.

Način na koji se neuroni međusobno povezuju određuje se arhitekturom neuronske mreže [26]. Jednoslojna neuronska mreža, tzv. perceptron, bila je prva neuronska mreža koja se sastojala od jednog sloja neurona (izlaznog sloja). Glavni nedostatak perceptrona bio je nemogućnost klasificiranja skupova podataka koji nisu linearno odvojivi. Kao rješenje tog problema razvijen je višeslojni perceptron koji uz ulazni i izlazni sloj sadrži jedan ili više skrivenih slojeva. Skriveni sloj sadržava transformirane ulaze koji su linearno odvojivi nakon čega se nelinearnost ostvaruje ponovnim postupkom ponderiranja i prolaska kroz aktivacijsku funkciju. Višeslojni perceptron s jednim skrivenim slojem prikazan je slikom 5.11.



Slika 5.11. Prikaz višeslojnog perceptrona s jednim skrivenim slojem [27]

Veći broj skrivenih slojeva rezultirati će sve složenijim nelinearnim modelom. Učenje pod nadzorom kod višeslojnog perceptrona (MLP) odvija se pomoću metode unazadne propagacije. Unazadna propagacija najčešće je korištena kod optimizacijskog algoritma gradijentnog spusta kako bi prilagodila težine neurona računajući gradijent i funkciju pogreške [9]. U početku učenja vrijednosti težinskih faktora su nasumično izabrane. Nakon dobivenog predviđanja za svaku podatkovnu točku izračuna se gubitak, npr. metodom unakrsne entropije. Ažuriranje težina vrši uz pomoć funkcije pogreške i izvođenja gradijentnog spusta kako je ranije opisano u poglavlju logističke regresije. Pogreška se zatim propagira na početak umjetnog neurona ili neuronske mreže i pridodaje se težinskim faktorima [25]. Razlika MLP-a u odnosu na logističku regresiju stoga je u tome što između ulaznog i izlaznog sloja postoji jedan ili više nelinearnih, skrivenih slojeva.

U cilju pronalaženja optimalnih rješenja odabir odgovarajućeg broja skrivenih slojeva i neurona uzrokuje složenost modela. Prilikom ugađanja hiperparametara unutar Python-a omogućuje se da stopa učenja bude definirana na tri raličita načina. Uz opcije konstantnog održavanja i postupnog smanjivanja stope učenja u svakom koraku iteracije nudi se mogućnost odabira adaptivne metode. Opcijom *adaptive* stopa učenja održava se konstantnom sve dok se gubitak, tj. pogreška smanuje, no u slučaju da se unutar dvije uzastopne iteracije pogreška ne umanji barem za vrijednost tolerancije, stopa učenja podešava se na pet puta manju vrijednost. Ova prilagodba od koristi je jedino u slučaju SGD

optimizatora, budući da lbfgs, koji odgovara podvrsti Newtonove metode, ne sadrži stopu učenja, a Adam optimizator uvijek koristi prilagodljivu stopu učenja. Ostali hiperparametri podešavaju se na isti način kao u slučaju prethodnih modela.

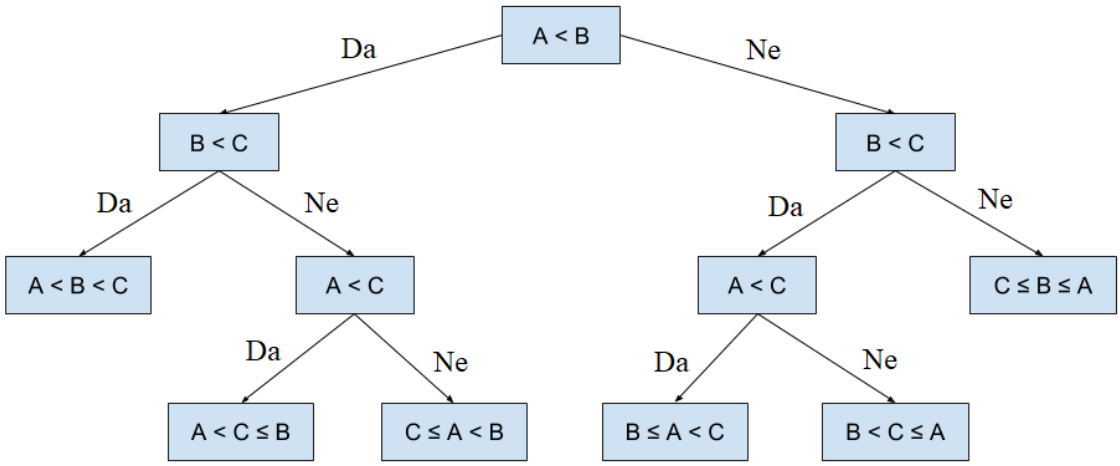
```
def MLPParRandomSearch():
    parameters = []
    #Choosing number of hidden layers and neurons
    def hidLayerSize():
        numHidLayers = random.randint(2,5)
        HLS = []
        for i in range(numHidLayers):
            HLS.append(random.randint(10,100))
        return tuple(HLS)
    #Choosing activation function
    ActFun = random.choice(['identity', 'logistic', 'tanh', 'relu'])
    #Choosing solver
    Solver = random.choice(['lbfgs', 'sgd', 'adam'])
    #Choosing strength of the L2 regularization term
    Alpha = random.uniform(1e-6, 1e-2)
    #Choosing learning rate schedule
    LearnRate = random.choice(['constant', 'invscaling', 'adaptive'])
    #Choosing maximum number of iterations
    MaxIter = random.randint(200,2000)
    #Choosing Tolerance
    Tol = random.uniform(1e-10, 1e-4)
    #Maximum number of iterations without change
    while True:
        nIter = random.randint(10,10000)
        if nIter < MaxIter:
            print("nIter smaller than MaxIter")
            break
        else:
            continue
    ##############################################
    #Appending Randomly Selected Data into Parameters list
    ##############################################
    parameters.append(hidLayerSize())
    parameters.append(ActFun)
    parameters.append(Solver)
    parameters.append(Alpha)
    parameters.append(LearnRate)
    parameters.append(MaxIter)
    parameters.append(Tol)
    parameters.append(nIter)

    return parameters
```

5.5. Klasifikator stabla odlučivanja i ekstremno randomizirano stablo

Stablo odlučivanja (eng. Decision Tree) neparametarski je model nadziranog učenja koji zbog svoje jednostavnosti i visoke točnosti spada u najpopularnije klasifikacijske algoritme. Karakteristika jednostavnosti proizlazi iz toga što način na koji dodjeljuje odgovarajuću klasu

je lako razumljiv, budući da je vrlo sličan ljudskom načinu razmišljanja prilikom donošenja pojedine odluke. Analogno tomu, algoritam na osnovu značajki skupa podataka postavlja niz osmišljenih pitanja nebi li sakupio što više informacija potrebnih za ispravno klasificiranje određenog primjera. Postavljanjem pitanja skup podataka se neprestano dijeli u manje podskupove na temelju testa vrijednosti značajke. Proces dijeljenja se ponavlja sve dok podatkovne točke unutar podskupova nemaju istu vrijednost ciljne varijable. Na taj način podaci se organiziraju u strukturu nalik stablu. Stablo se sastoji od niza čvorova i grana, gdje čvorovi označavaju test značajki, a grane predstavljaju ishod testa, odnosno da/ne rezultat. Prvi test ispituje se tzv. korijenskim čvorom, čiji ishodi zatim dijele skup podataka na temelju vrijednosti značajke, čime se stvaraju novi, interni čvorovi. Dolaskom do čvorova u kojima podatkovne točke odgovaraju istoj vrijednosti ciljne varijable prekida se testiranje značajki, nema daljnjih grananja te se takvi čvorovi nazivaju lisnim čvorovima. Svaki lisni čvor dodijeljuje se jednoj klasi koja predstavlja najprikladniju ciljanu vrijednost [28]. Jednostavan primjer stabla odlučivanja prikazan je slikom 5.12.



Slika 5.12. Primjer stabla odlučivanja [29]

Na slici 5.12 prikazan je jednostavan primjer odlučivanja u kakvom su odnosu vrijednosti A, B i C na način da se pojedinačno ispituje veza između njih. Nakon što se konstruira stablo odlučivanja klasificiranje testnih primjera je vrlo jednostavno. Počevši od korijenskog čvora primjenjuje se testni uvjet na zapis, nakon čega se prati odgovarajuća grana na temelju ishoda testa. Ishod će zatim usmjeriti prema drugom unutarnjem čvoru, za koji se primjenjuje novi testni uvjet, ili prema lisnom čvoru koji može poprimiti jedno od 6 različitih rješenja, tj. klasa.

Općenito, prilikom konstruiranja stabla odlučivanja potrebno je razriješiti dva glavna problema; kako bi se podaci o obuci trebali podijeliti i kada bi se proces dijeljenja trebao zaustaviti [30]. Problem podjele podataka odnosi se na odabir optimalne značajke za korijenski čvor i za interne čvorove. Značajka koja će se koristiti kao uvijet testiranja određuje se na osnovu vrijednosti tzv. mjere nečistoće. Iako se ranije spomenulo kako će se stablo granati sve dok podatkovne točke unutar lisnih čvorova ne budu pripadale istoj klasi, takav slučaj u strojnom učenju nije poželjan, budući da to lako rezultira overfittingom. Stoga se u većini slučajeva dio lisnih čvorova unutar stabla odlučivanja smatra "nečistim", što znači da je nekolicina primjera unutar čvora dodijeljena pogrešnoj klasi. Uz pomoć različitih mjera lako se može izračunati nečistoća pojedinog čvora, pri čemu je naravno poželjno da vrijednost bude što manja. Shodno tomu, za svako grananje biti će odabrana značajka čije će testiranje rezultirati čvorovima s najmanjim nečistoćama. Dvije glavne mjere nečistoće koje se koriste kod stabla odlučivanja su entropija i Gini nečistoća. Rezultat Ginijevog indeksa označava vjerojatnost da je slučajno odabrana podatkovna točka unutar čvora pogrešno klasificirana. Može poprimiti vrijednost između 0 i 1, pri čemu će niža vrijednost odgovarati manjoj nečistoći. Ginijev indeks računa se formulom:

$$I_G = 1 - \sum_{i=1}^{c} p_i^2 \,, \tag{5.15}$$

gdje je:

I_G – Ginijev indeks,

p_i – udio primjera koji pripadaju i-toj klasi i

c – broj klasa.

Na sličan način mjeri se nečistoća primjenom entropije. Niža vrijednost također odgovara manjoj nečistoći, a maksimalna vrijednost koja se može ostvariti ovisi o broju klasa. Iako u većini slučajeva nema značajne razlike u dobivenim rezultatima, Gini nečistoća se češće koristi kod velikih skupova podataka, s obzirom na kompleksniji izračun entropije čija formula glasi:

$$E = -\sum_{i=1}^{c} p_i \log_2(p_i) \,, \tag{5.16}$$

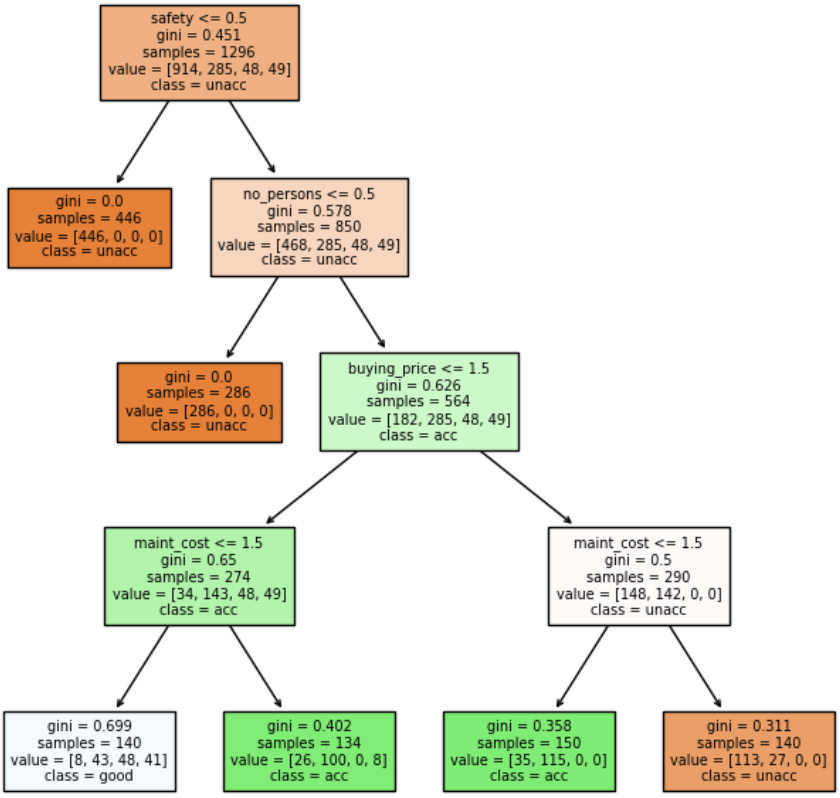
gdje je:

E – vrijednost entropije,

p_i – udio primjera koji pripadaju i-toj klasi i

c – broj klasa.

Što se tiče trenutka zaustavljanja grananja, odnosno dubine stabla odlučivanja, bitno je znati da preveliko stablo povećava mogućnost pojave overfittinga, dok malo stablo vrlo vjerojatno rezultira većom količinom nečistoće unutar lisnih čvorova, stoga je potrebno naći kompromis prilikom definiranja dubine stabla. Na slici 5.13. prikazano je stablo odlučivanja za analizirani skup podataka, pri čemu je radi lakšeg razmatranja dubina definirana na vrijednost 4, što znači da stablo sadrži 4 grananja.



Slika 5.13. Prikaz stabla odlučivanja "car evaluation" skupa podataka

Kao što se može vidjeti na slici 5.13, uz značajke koje se testiraju, čvorovi stabla sadržavaju i informacije o vrijednosti Gini indeksa, broju primjera (podatkovnih točaka), broju primjera po klasama i klasi kojoj je pripada najveći broj primjera. Grane čvorova koje idu u lijevu stranu označavaju pozitivan ishod testa značajke, dok grane prema desnoj strani označavaju

negativan ishod. Unosom vrijednosti u jednadžbu 5.14 za svaki čvor se vrlo lako se može potvrditi iznos Gini indeksa. Isto tako, promatrajući raspored čvorova može se zaključiti kako je redoslijed testiranja pojedine značajke ispravno odabran, budući da se od ranije zna kako su automobili s niskom razinom sigurnosti te automobili u koje stanu samo dvije osobe potpuno neprihvatljivi, stoga je razumljivo da se te dvije značajke prve ispituju. Na taj način algoritam je već nakon prva dva grananja prepolovio broj primjera koje mora ispravno klasificirati. Usporedba značajki s brojčanim vrijednostima kod svakog čvora posljedica je mapiranja vrijednosti značajki.

Kako bi se zadržala kvaliteta rezultata, a istodobno umanjila mogućnost prevelike složenosti stabla kojeg bi bilo teško tumačiti, prilikom definiranja vrijednosti hiperparametara unutar Python-a moguće je prilagođavati maksimalan broj lisnih čvorova kao i najmanji broj uzoraka koji je potreban da bi čvor stekao ulogu lisnog čvora. Uz to, moguće je i odabrati metodu koja će se koristi za odabir podjele na svakom čvoru, tj. način odabira vrijednosti značajke koja će poslužiti kao uvjet za ispitivanje. Pritom se nude opcije *best* i *random*, gdje *best* označava odabir najbolje podjele, a *random* odabir najbolje slučajne podjele.

```
def DecTreeParRandomSearch():
    parameters = []
    #Function to measure the quality of a split
    Criterion = random.choice(['gini', 'entropy'])
    #Strategy used to choose the split at each node
    Splitter = random.choice(['best', 'random'])
    #Maximum depth of the tree
    MaxDepth = random.randint(8,18)
    #Minimum number of samples required to be at leaf node
    MinSamplLeaf = random.randint(1,10)
    #Maximum number of leaf nodes
    MaxLeafNodes = random.choice([None, 40,50,60,70,80,90])
    ################################################
    #Appending Randomly Selected Data into Parameters list
    ################################################
    parameters.append(Criterion)
    parameters.append(Splitter)
    parameters.append(MaxDepth)
    parameters.append(MinSamplLeaf)
    parameters.append(MaxLeafNodes)

    return parameters
```

Ekstremno randomizirano stablo (eng. Extremely Randomized Tree) inačica je klasičnog stabla odlučivanja čija se konstrukcija temelji na potpuno nasumičnom odabiru vrijednosti značajki koje služe kao uvjet ispitivanja. Za razliku od stabla odlučivanja ekstremno randomizirano stablo prilikom odabira značajke za iduće grananje nikad ne računa mjeru

nečistoće za svaku značajku. Težeći ka optimalnom odabiru postavki prilikom grananja stablo odlučivanja zna biti sklono pretjeranom opremanju, stoga ovaj algoritam može ponekad biti bolja opcija, iako u većini slučajeva rezultira većom dubinom stabla. Naravno, minimalnom izmjenom vrijednosti hiperparametara unutar Python-a omogućeno je oba modela mogu poprimiti karakteristike suprotnog algoritma i tako generirati identične rezultate.

5.6. Prikaz i usporedba rezultata

Kao što je ranije spomenuto, za svaki se model realizira ista vrsta petlje unutar koje se poziva funkcija nasumičnog odabira vrijednosti hiperparametara nakon čega započinje treniranje i testiranje modela. Izvršavanje petlje traje sve dok ROC AUC rezultat ne dosegne vrijednost koja je unaprijed definirana uvjetom (npr. 0.999) nakon čega se dobiveni rezultat i kombinacija hiperparametara pohranjuju kao konačna rješenja. U protivnom, petlja će se nastojati izvršiti 1000 puta i na taj način generirati 1000 raličitih kombinacija, nakon čega se kao konačno rješenje memorira kombinacija kojom se postigao najbolji ROC AUC rezultat.

```
k = 0
BestScore = 0
BestParam = []

while True:
    print("Current Iteration = {}".format(k))
    Param = ModelParRandomSearch()
    test = Model(Param,X_train,y_train, X_test, y_test)
    k+=1
    if (test > 0.999):
        BestScore = test
        print("Solution is Found!!")
        break
    elif (test > BestScore):
        BestScore = test
        BestParam = Param
    elif (k >= 1000):
        print("Algorithm has reached maximum number of iterations!!\n")
        print("Best score = {}".format(BestScore))
        print("Best parameters = {}".format(BestParam))
        break
    else:
        continue
```

Tablica 5.1. Optimalna kombinacija hiperparametara za klasifikator Logistic Regression

Kazna	Tolerancija	C	Rješavač	Max. br. iter.
None	8.446e-07	0.105	saga	156

Tablica 5.2. Optimalna kombinacija hiperparametara za SGD klasifikator

Funkcija gubitka	Kazna	Alfa	Max. br. iter.	Tolerancija	Br. iter. bez promjene
modified huber	L1	0.015	1119	4.39e-04	182

Tablica 5.3. Optimalna kombinacija hiperparametara za KNN klasifikator

Br. susjeda	Težinska funkcija	Algoritam
11	uniform	brute

Tablica 5.4. Optimalna kombinacija hiperparametara za klasifikator Radius Neighbors

Radius	Težinska funkcija	Algoritam
1.027	uniform	auto

Tablica 5.5. Optimalna kombinacija hiperparametara za MLP klasifikator

Veličina skrivenih slojeva	Aktivacijska funkcija	Rješavač	Alfa	Stopa učenja	Max. br. iter.	Tolerancija	Br. iter. bez promjene
(82,31)	logistic	lbfgs	0.007	adaptive	1375	6.732e-05	1272

Tablica 5.6. Optimalna kombinacija hiperparametara za klasifikator Decision Tree

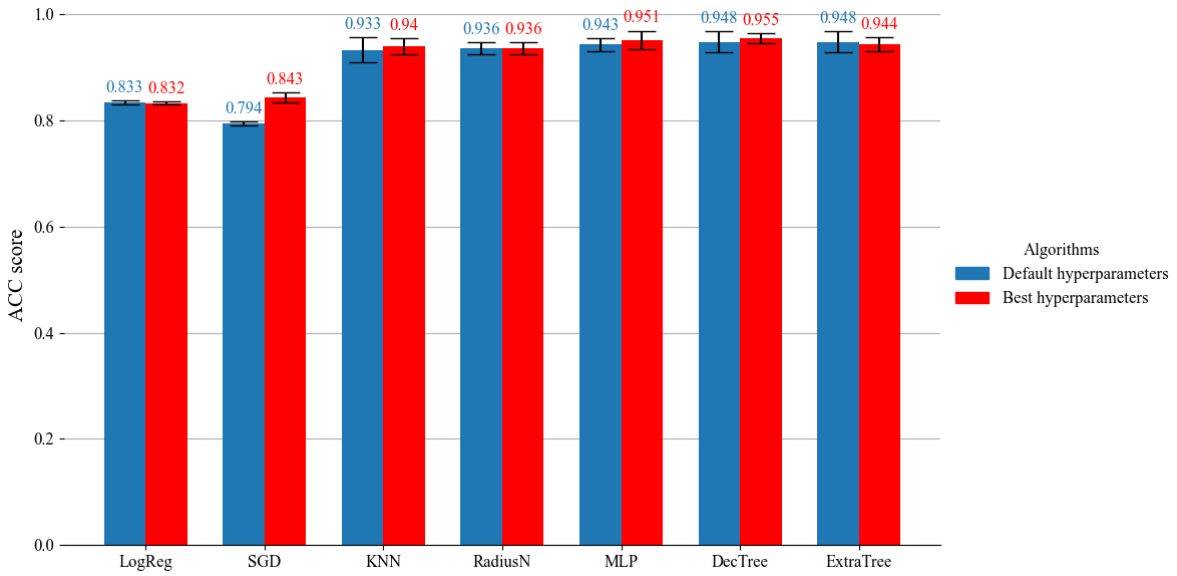
Kriterij	**Metoda podjele**	**Max. dubina stabla**	**Min. br. uzoraka za lisni čvor**	**Max. br. lisnih čvorova**
gini	random	11	8	None

Tablica 5.7. Optimalna kombinacija hiperparametara za klasifikator Extra Tree

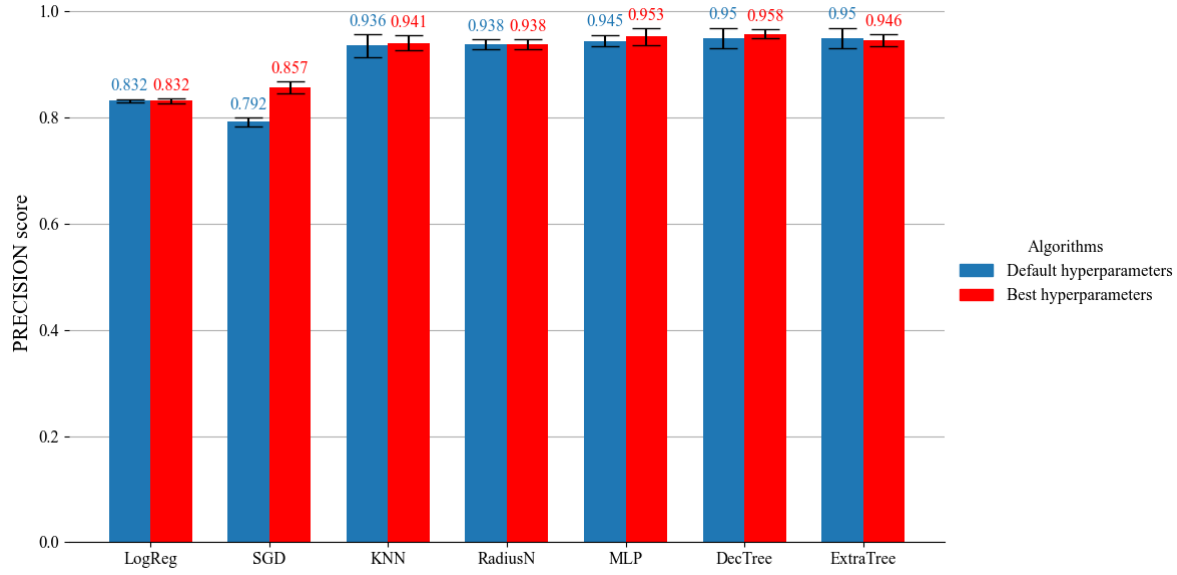
Kriterij	**Max. dubina stabla**	**Min. br. uzoraka za lisni čvor**	**Max. br. lisnih čvorova**
entropy	19	4	100

Dobivene rezultate sada je moguće usporediti s rezultatima ostvarenim preko defaultnih vrijednosti hiperparametara. Za to se koristi funkcija koja za svaki model kreira dva stupčasta dijagrama na osnovu kojih se dobiva uvid u razlike među vrijednostima metrika za pojedine klasifikatore. Prikazi usporedbe rezultata pojedinih metrika predočeni su slikama u nastavku.

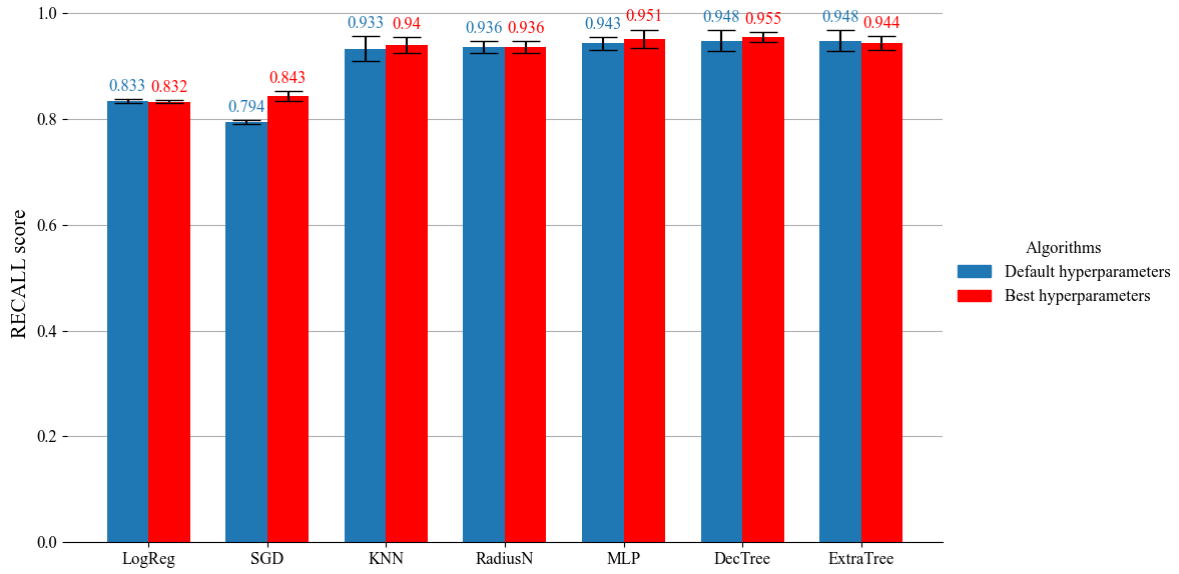
```
def PlotGraph(metricMeanScore, metricBestMeanScore, ylab):
    plt.figure(figsize=(12,6), layout='tight')
    x_axis = np.arange(len(algorithms))*4
    roundScore = [round(elem, 3) for elem in metricMeanScore]
    roundBestScore = [round(elem, 3) for elem in metricBestMeanScore]
    bar1 = plt.bar(x_axis -0.7, roundScore, width = 1.4)
    bar2 = plt.bar(x_axis +0.7, roundBestScore, width = 1.4, color='r')
    plt.xticks(x_axis, algorithms)
    plt.ylabel(ylab, fontsize=15)
    color = ('tab:blue', 'r')
    cmap = dict(zip(color,[bar1, bar2]))
    patches = [Patch(color=v, label=k) for v,k in cmap.items()]
    plt.legend(title='Algorithms', labels=['Default parameters', 'Best parameters'],
            handles=patches, bbox_to_anchor=(1, 0.5), loc='center left', frameon=False)
    plt.bar_label(bar1, padding=3, color='tab:blue')
    plt.bar_label(bar2, padding=3, color='r')
    plt.rcParams['axes.spines.left'] = False
    plt.rcParams['axes.spines.right'] = False
    plt.rcParams['axes.spines.top'] = False
    plt.rcParams['axes.spines.bottom'] = True
    plt.grid(axis = 'y')
    plt.rc('axes', axisbelow=True)
    plt.show()
```



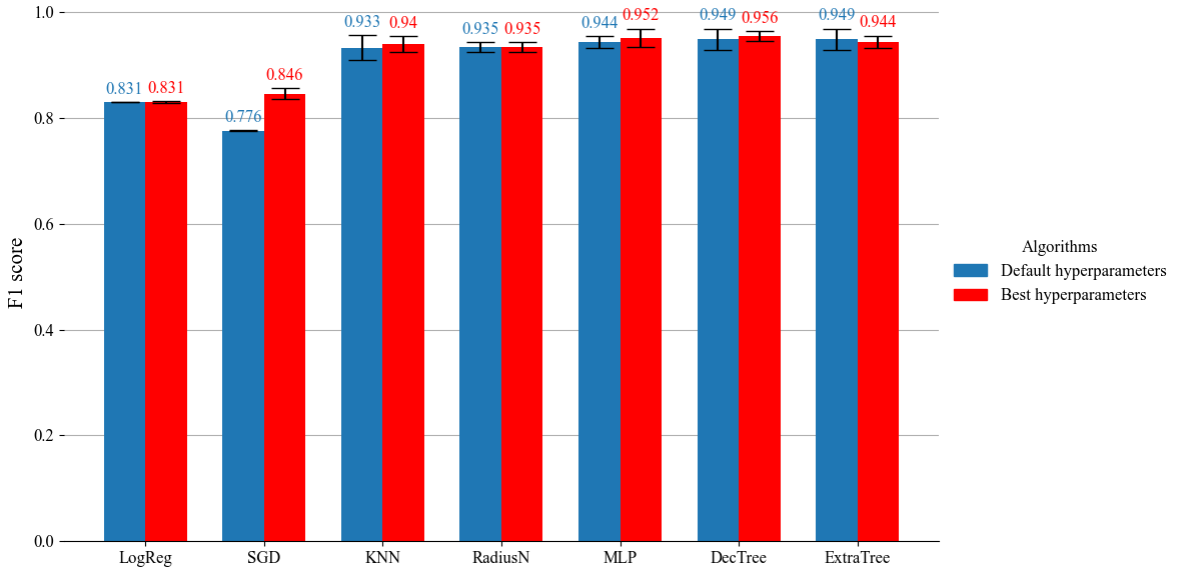
Slika 5.14. Prikaz usporedbe rezultata točnosti s defaultnim i najboljim vrijednostima hiperparametara



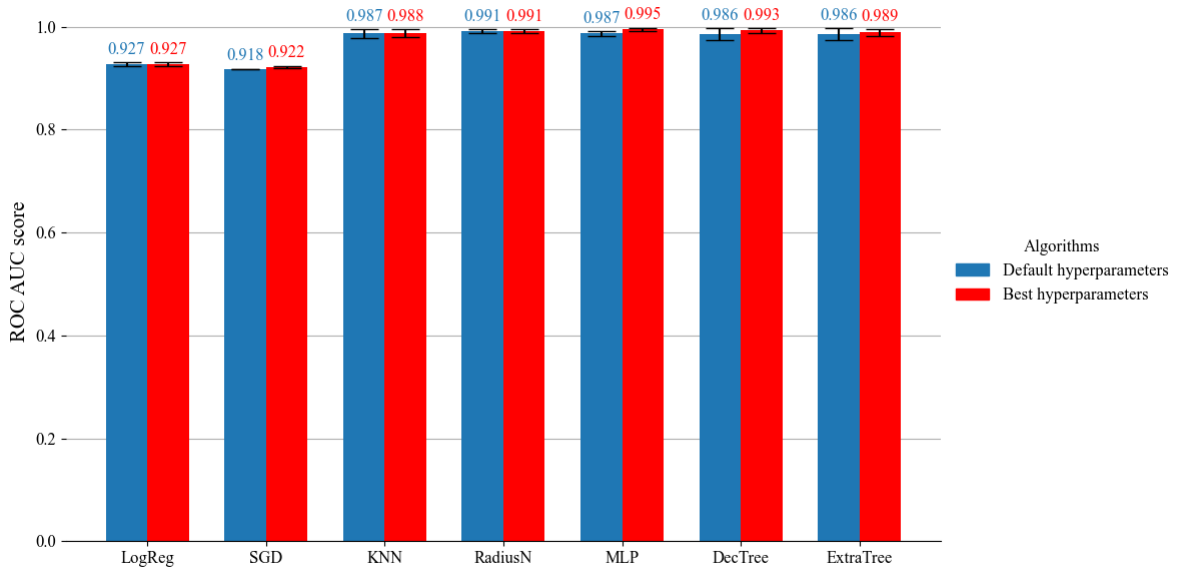
Slika 5.15. Prikaz usporedbe rezultata preciznosti s defaultnim i najboljim vrijednostima hiperparametara



Slika 5.16. Prikaz usporedbe rezultata opoziva s defaultnim i najboljim vrijednostima hiperparametara



Slika 5.17. Prikaz usporedbe F1 rezultata s defaultnim i najboljim vrijednostima hiperparametara



Slika 5.18. Prikaz usporedbe ROC AUC rezultata s defaultnim i najboljim vrijednostima hiperparametara

Budući da kod većine algoritama postoji relativno mali prostor rješenja koji bi se trebao premašiti da bi se postigao optimalan rezultat, uzeći u obzir da je rješenje od 99.9% gotovo nemoguće doseći, svaki ostvareni postotak je od koristi. Iz grafova se može primjetiti kako su SGD, MLP i Decision Tree algoritam u prosjeku postigli najveći napredak u odnosu na rezultate dobivene preko defaultnih vrijednosti hiperparametara, dok su općenito najbolje rezultate metrika postigli KNN, Radius neighors, MLP, Decision Tree i Extra Tree klasifikator. Povećanje vrijednosti ROC AUC rezultata ne mora nužno značiti da će se vrijednosti preostalih metrika povećati, dapače mogu se i smanjiti. Nažalost, s obzirom da je kao kriterij odabira najboljih vrijednosti hiperparametara poslužio samo ROC AUC rezultat, koji je kod svakog algoritma uspio ostvariti pozitivnu promjenu, neki klasifikatori, kao što je Extra Tree klasifikator, su kod ostalih metrika ostvarili negativne promjene.

6. UNAKRSNA VALIDACIJA I METODA ANSAMBLA

U ovom poglavlju opisati će se metoda nadogradnje klasične train/test podjele s kojom se ostvaruje stabilnija procjena izvedbe modela koji se nastoji istrenirati. Riječ je o tzv. unakrsnoj validaciji, koja će pritom za izračun rezultata metrika koristiti metodu k-struke. Klasifikatori koji će unakrsnom validacijom ostvariti najbolja rješenja zatim će se primjeniti za formiranje ansambl modela.

6.1. Unakrsna validacija

Kao što se može primijetiti iz prethodnih poglavlja metoda podjele skupa podataka na skupove za treniranje i testiranje lako je razumljiva i stoga predstavlja najjednostavniji sistem predviđanja za primjere s kojima se klasifikator nikad nije susreo. Ipak, uz jednostavnost dolaze i određeni nedostaci, stoga ova metoda neće uvijek biti od koristi. Metoda train/test podjele unosi nestabilnost prilikom procjene izvedbe modela. Naime, u realnom svijetu, informacija o tome hoće li model koji je istreniran na određenom skupu podataka točno klasificirati nepoznate primjere nije poznata. Kao rješenje tog problema nudi se metoda unakrsne provjere valjanosti. Unakrsna provjera valjanosti ili unakrsna validacija (eng. Cross Validation (CV)) je metoda koja višestrukom podjelom skupa podataka nastoji testirati model na nasumičnim skupovima podataka. Jednostavno rečeno riječ je o izvršavanju train/test metode određeni broj puta na podacima s kojima se model trenira. Stoga, cilj unakrsne validacije je dati procjenu koliko dobro model funkcionira na neviđenim podacima. Na taj način dobiva se stabilnija procjena izvedbe modela u odnosu na klasičnu train/test medodu.

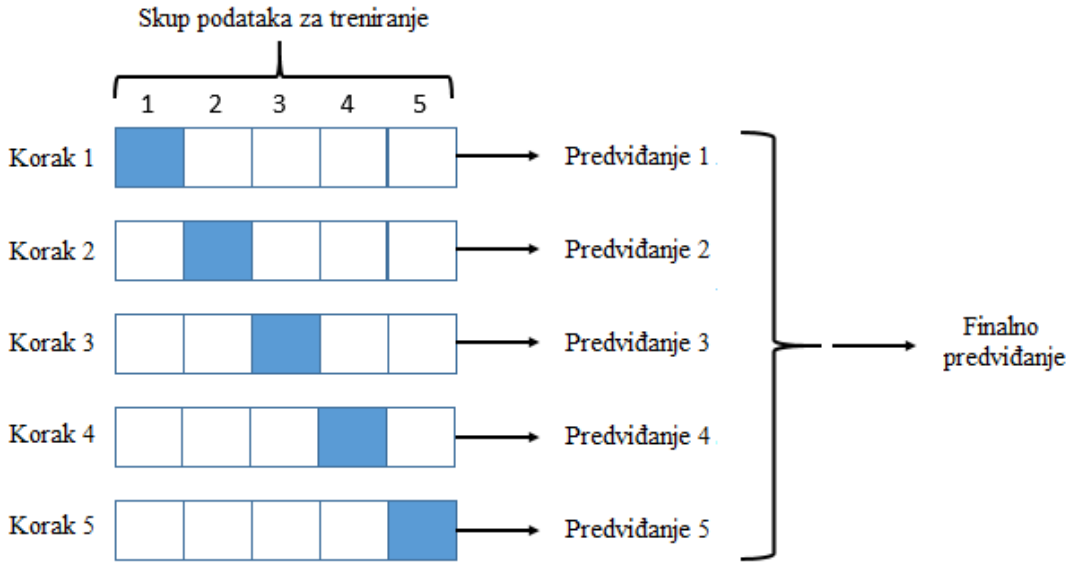
Izvođenje CV-a temelji se na podjeli skupa podataka za obuku u podskupove, gdje jedan podskup ili uzorak služi za testiranje, a ostatak za treniranje. Proces se ponavlja sve dok svaki podskup skupa za obuku ne bude iskorišten za testiranje. Posljedica toga jest generiranje različitih podmodela čiji broj ovisi o broju podskupova na koje se skup za trening podijelio. Rezultat koji zatim opisuje kvalitetu korištenog algoritma odgovara srednjoj vrijednosti rezultata podmodela. Pritom postoje različite vrste CV-a, a neke od najpoznatijih su:

- k-struka unakrsna validacija,
- stratificirana k-struka unakrsna validacija i
- unakrsna validacija "izdvoji jednog".

K-struka unakrsna validacija (eng. K-fold Cross Validation) predstavlja najjednostavniji koncept unakrsne validacije u kojoj se skup podataka dijeli na *k* jednakih podskupova i na taj način generira *k* različitih podmodela. Njegova nadogradnja odgovara tzv. stratificiranoj unakrsnoj validaciji (eng. Stratified Cross Validation) koja osigurava da svaki od *k* podskupova sadrži približno isti postotak uzoraka svake ciljne klase kao i cijeli skup [31], zbog čega je prilično korisna u slučaju kada skup podataka nije uravnotežen. Unakrsna validacija "izdvoji jednog" (eng. Leave One Out Cross Validation (LOOCV)) metoda je u kojoj se testiranje provodi samo na jednom uzorku. Iako to rezultira još preciznijom procjenom modela, zbog potrebe za većom računalnom snagom i dužim periodom izvođenja rijetko se koristi kod velikih skupova podataka. Za potrebe ovog rada implementirati će se k-struka unakrsna validacija za vrijednosti $k = 5$ i $k = 10$.

6.1.1. K-struka unakrsna validacija

K-struka unakrsna validacija je najosnovnija i najčešće korištena metoda na čijem konceptu su bazirani preostali pristupi izvođenja CV-a. Metoda funkcionira na način da se skup za obuku podijeli na *k* jednaka podskupa, tzv. preklopa, nakon čega započinje proces izvođenja *k* iteracija treniranja i testiranja tako da se unutar svake iteracije različit podskup koristi za testiranje, dok se preostalih k-1 preklopa koristi za treniranje. Proces sa 5 preklopa je ilustriran slikom 6.1.



Slika 6.1. Prikaz izvođenja k-struke unakrsne validacije za k = 5 [32]

Po završetku posljednjeg koraka odrađuje se prosjek izvedbe modela u odnosu na svaki preklop, nakon čega se dobiva finalna procjena izvedbe modela, kao što je prikazano na slici 6.1. Ovaj je pristup računalno zahtjevniji u odnosu na klasičnu train/test metodu jer zahtijeva obuku i evaluaciju modela *k* puta, međutim rezultirajuća procjena je pouzdanija [33]. Odabir broja preklopa pritom je od velike važnosti, budući da loše odabrana vrijednost za *k* može rezultirati pogrešnom procjenom izvedbe modela. 5-struka i 10-struka unakrsna validacija najčešće su korištene *k* vrijednosti za procjenu modela strojnog učenja [33], što bi značilo da se u većini slučajeva prilikom izvođena procesa CV-a u svakom koraku 80%, odnosno 90% podataka koristi za treniranje.

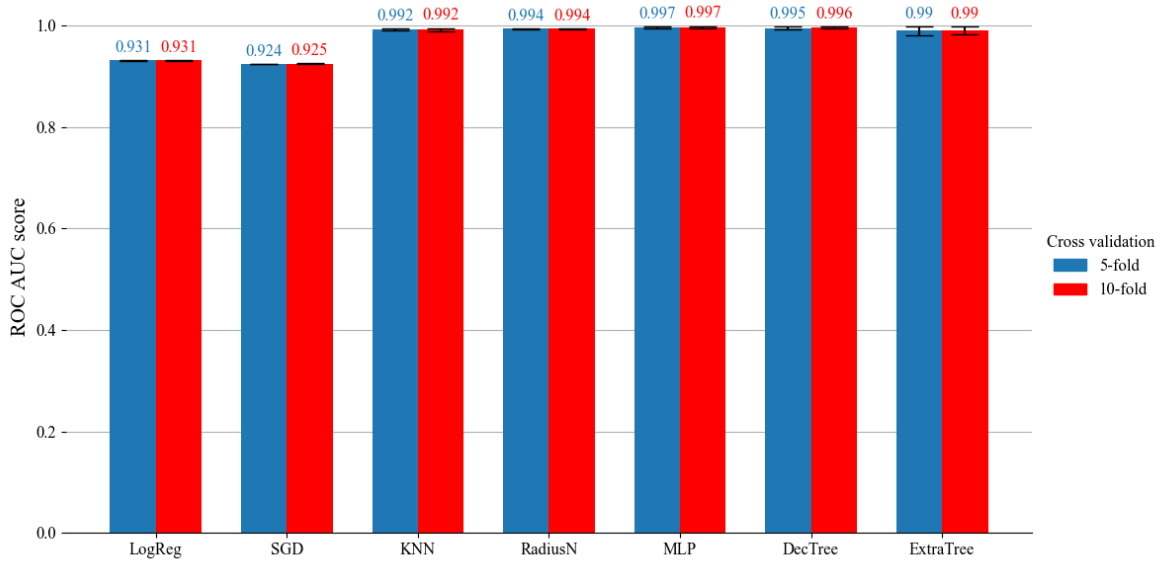
Unutar programskog jezika Python unakrsna validacija izvodi se korištenjem funkcije `cross_validate` u koju je potrebno unijeti osnovne elemente, kao što su model, skup podataka za treniranje, broj preklopa i metrika ili metrike koje će poslužiti za procjenu izvedbe. Također, moguće je ostvariti i izračun rezultata metrika na skupu za obuku u svakom koraku, što je korisno u slučaju da je cilj dobiti prosječan rezultat treniranja i testiranja.

```
cvmodel = cross_validate(model, X_train, y_train, cv=5,
                         scoring = "roc_auc_ovr_weighted",
                         return_train_score=True)

AvrROCAUCScoreTrain = np.mean(cvmodel['train_roc_auc_ovr_weighted'])
StdROCAUCScoreTrain = np.std(cvmodel['train_roc_auc_ovr_weighted'])
AvrROCAUCScoreTest = np.mean(cvmodel['test_roc_auc_ovr_weighted'])
StdROCAUCScoreTest = np.std(cvmodel['test_roc_auc_ovr_weighted'])
AvrAllROCAUCScore = np.mean([AvrROCAUCScoreTrain, AvrROCAUCScoreTest])
StdAllROCAUCScore = np.std([AvrROCAUCScoreTrain, AvrROCAUCScoreTest])
```

Tablica 6.1. ROC AUC rezultati i standardne devijacije modela dobiveni 5-fold i 10-fold unakrsnom validacijom

Model	**ROC AUC (CV5)**	**ROC AUC Std (CV5)**	**ROC AUC (CV10)**	**ROC AUC Std (CV10)**
Logistic Regression	0.93082	9.27e-04	0.93113	5.105e-04
SGD	0.92417	3.674e-04	0.92454	4.682e-04
KNN	0.99161	22.701e-04	0.99159	31.445e-04
Radius Neighbors	0.99371	7.236e-04	0.99386	7.604e-04
MLP	0.9969	17.89e-04	0.99662	19.082e-04
Decision Tree	0.9953	26.12e-04	0.99609	15.966e-04
Extra Tree	0.98966	84.959e-04	0.98966	80.627e-04



Slika 6.2. Prikaz usporedbe ROC AUC rezultata modela dobivenih 5-fold i 10-fold unakrsnom validacijom

Tablicom 6.1 i slikom 6.2 predočene su najviše vrijednosti ROC AUC rezultata te standardne devijacije za 5-struku i 10-struku unakrsnu validaciju dobivene nakon ponovnog izvođenja petlje kojom se generiralo 1000 različitih kombinacija hiperparametara. MLP klasifikator je i u ovom slučaju postigao najbolji rezultat, dok su linearni klasifikatori (logistička regresija i SGD) ponovo ostvarili nešto lošije rezultate u odnosu na ostale modele. S obzirom na neznatne razlike u rezultatima za različitu k-struku, zaključuje se kako povećanje broja preklopa u ovom slučaju neće biti učinkovito, budući da se minimalna poboljšanja dobivaju na račun znatnog povećanja računalne snage i vremena izvođenja. Što se tiče vrijednosti standardne devijacije, Extra Tree klasifikator je zbog svoje složenosti, koja je posljedica povećane dubine stabla, postigao visoku vrijedost na podacima za treniranje i relativno nisku vrijednost na podacima za testiranje, što je za rezultat dalo znatno lošiju standardnu devijaciju u odnosu na ostale modele.

6.2. Metoda ansambla

Ansambl učenje (eng. Ensemble learning) ili skupno učenje metoda je koja se temenji na kombinaciji predviđanja više modela strojnog učenja radi poboljšanja ukupne izvedbe. Pretpostavke o skupnom učenju slijede ideju kolektivne mudrosti da je više glava općenito bolje od jedne [34]. Drugim riječima, pretpostavka je da će prosječno predviđanje od strane

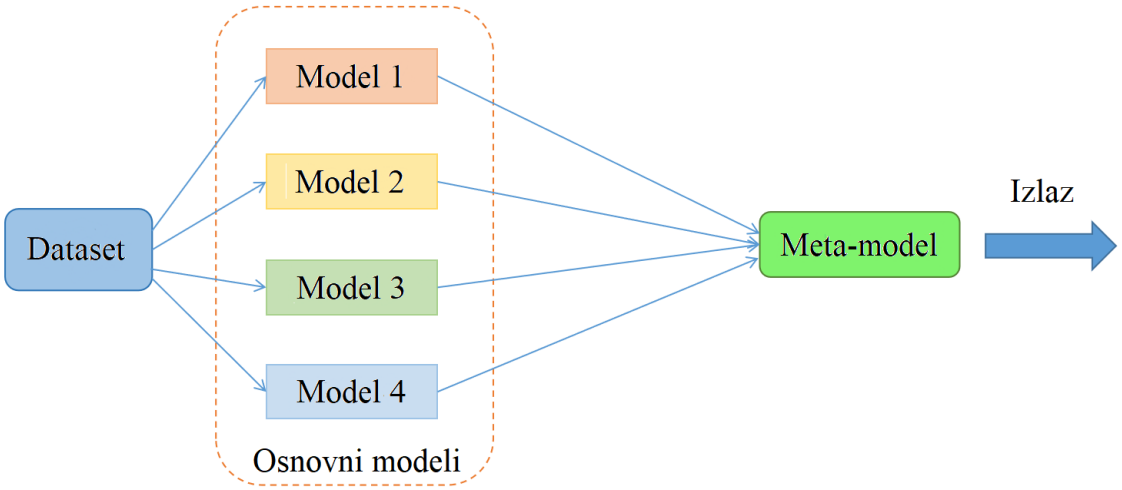
većeg broja ljudi biti točnije naspram predviđanja od strane jednog čovjeka. Jedna teorijska osnova za kolektivnu mudrost može se izvesti iz Condorcetovog teorema porote koji se odnosi na porotu od *n* glasača koji donose odluku, pri čemu je pretpostavka da postoje dvije opcije prilikom glasanja i da je vjerojatnost odabira ispravne odluke *p* jednaka kod svakog glasača. Condorcetov teorem o poroti tada kaže da, u slučaju kompetentnosti glasača ($p > 0.5$), vjerojatnost ispravne odluke teži ka jedinici kako se broj birača povećava prema beskonačnosti [34]. Ovaj način razmišljanja od koristi je samo u slučaju postojanja raznolikosti među korištenim algoritmima. Naime, različiti algoritmi ne griješe na isti način stoga se kombiniranjem nastoje kompenzirati nedostaci pojedinih algoritama. U slučaju nepostojanja različitosti među algoritmima, njihovom kombinacijom nemože se postići bolji rezultat u odnosu na rezultat pojedinog algoritma.

Iako postoji više vrsta ansambla tri tehnike koje dominiraju i koje se najčešće upotrebljavaju su metode:

- Bagging,
- Boosting i
- Stacking.

Metoda pakiranja (eng. Bagging) je tehnika poboljšanja izvedbe koja svoj proces predviđanja dijeli u dvije faze, zbog čega se još naziva Bootstrap agregacija (eng. Bootstrap Aggregating). Bootstrapping je faza u kojoj se svaki klasifikator trenira na podskupu čiji uzorci su izvedeni iz izvornog skupa metodom zamjene, što znači da pojedine instane mogu biti odabrane više puta. U fazi agregacije kombiniraju se rezultati pojedinih klasifikatora te se konačna procjena ostvaruje postupkom većinskog glasovanja. Atraktivno svojstvo strategije pakiranja je da smanjuje varijancu dok istovremeno zadržava pristranost koja pomaže u izbjegavanju prekomjernog opremanja [35]. Za razliku od Bagging metode, kod tehnike pojačavanja (eng. Boosting) modeli se pokreću sekvencijalno. Riječ je o kombinaciji nekoliko slabijih klasifikatora koji nastoje učiti iz pogrešaka svojih prethodnika. Umjesto generiranja svih podskupova u isto vrijeme, generira se jedan po jedan podskup koji se zatim koristi za obuku modela i testira se prema izvornim podacima [36]. Kako bi naredni model uspio ispraviti pogreške prošlog modela netočnim predviđanjima dodjeljuje se veća težina. Konačni model jakih performansi zatim se definira kao ponderirana srednja vrijednost svih slabih modela. Obje opisane metode mogu biti vrlo uspješne u svojoj implementaciji, no za potrebe ovog rada koristi se Stacking ansambl.

Slaganje (eng. Stacking) ili složena generalizacija (eng. Stacked generalization) je tehnika kombiniranja različitih modela u svrhu smanjenja njihovih pristranosti. Slično kao kod metode pakiranja, modeli se treniraju paralelno te imaju ulogu tzv. modela prve (osnovne) razine. No, za razliku od Bagging i Boosting ansambla, koji koriste stohastičke metode za kombiniranje osnovnih modela (učenika), Stacking ansambl koristi tzv. meta-model, tj. meta-klasifikator u slučaju problema klasifikacije. Primjer strukture Stacking ansambla prikazan je slikom 6.3.



Slika 6.3. Primjer strukture Stacking ansambla [37]

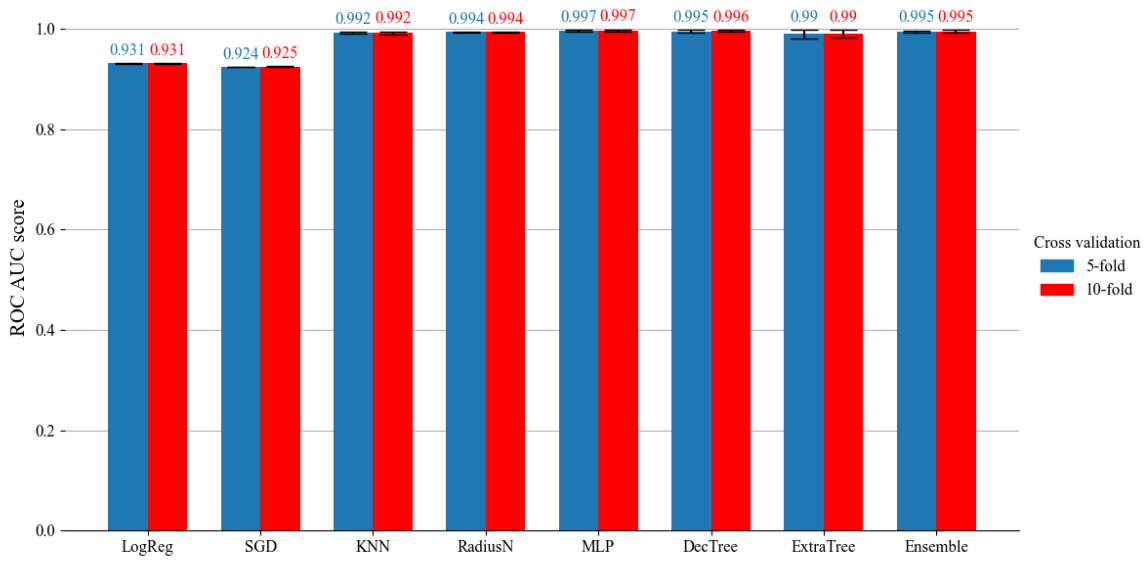
Osnovni modeli (modeli prve razine) pojedinačno se treniraju koristeći isti skup za vježbanje. Predviđanja generirana iz različitih modela strojnog učenja zatim se koriste kao ulazi u meta-model, odnosno model druge razine. Za razliku od drugih metoda ansambla, koje se koriste samo za kombiniranje predviđanja različitih modela, meta-model uči kako kombinirati predviđanja [36]. Poseban je to slučaj ponderiranog usrednjavanja (prosjeka) koji koristi skup predviđanja kao značajke i uvjetno odlučuje ponderirati ulazna predviđanja, što potencijalno rezultira boljom izvedbom [35]. Zahvaljujući uvježbanom meta-klasifikatoru na izlazu se zatim generira konačni rezultat.

Unutar Python-a implementacija Stacking ansambla za problem klasifikacije jednostavno se izvodi korištenjem funkcije `StackingClassifier`, pri čemu je potrebno unutar funkcije navesti klasifikatore koji će se kombinirati. Pritom se kao meta-klasifikator koristi logistička regresija, budući da je to meta-klasifikator koji se često koristi za slaganje [38]. S obzirom da

su linearni modeli ostvarili daleko najslabije rezultate prilikom izvođenja unakrsne validacije svi ostali klasifikatori koristiti će se za kreiranje ansambl modela.

Tablica 6.2. ROC AUC rezultati i standardna devijacije Stacking ansambla dobiveni 5-fold i 10-fold unakrsnom validacijom

ROC AUC (CV5)	ROC AUC Std (CV5)	ROC AUC (CV10)	ROC AUC Std (CV10)
0.99537	2.144e-03	0.99538	2.246e-03



Slika 6.4. Prikaz usporedbe ROC AUC rezultata pojedinačnih modela i ansambl modela dobivenih 5-fold i 10-fold unakrsnom validacijom

Iz tablice 6.2 i slike 6.4 može se primjetiti kako Stacking ansambl nije uspio nadvladati rezultat ostvaren MLP modelom. Time je utvrđeno kako je ansambl model vrlo osjetljiv što znači da bi za ostvarenje boljeg rezultata trebalo kombinirati algoritme koji mogu postići rezultate na razini MLP-a i algoritma stabla odlučivanja.

7. USPOREDBA REZULTATA

Radi usporedbe rezultata s rezultatima dobivenim u istraživačkim radovima [1] i [2] u nastavku je predočena tablica klasifikatora stabla odlučivanja i MLP klasifikatora, koji su se usto pokazali i kao najkvalitetniji modeli s obzirom na algoritme korištene u ovom radu. Naravno, budući da oba istraživačka rada učinkovitost algoritama ocjenjuju na osnovu njihove točnosti tablica 7.1 sadrži samo vrijednosti metrike točnosti koje su pritom postignute ugađanjem vrijednosti hiperparametara zahvaljujući tablicama 5.5 i 5.6.

Tablica 7.1. Vrijednosti metrike točnosti MLP i Decision Tree modela

	Točnost na skupu za treniranje (%)	**Točnost na testnom skupu (%)**	**Srednja točnost (%)**	**10-fold točnost (%)**
MLP	96.89	93.35	95.12	94.59
Decision Tree	95.59	95.38	95.5	92.68

Premda omjer podjele skupa na skupove za treniranje i testiranje, koji u ovom radu iznosi 80:20, ne odgovara niti jednom od omjera kojeg su ispitali autori u svojim radovima dobivene vrijednosti na testnom skupu su u većini slučajeva veće iako je optimalna kombinacija hiperparametara zapravo odabrana na osnovu najbolje vrijednosti ROC AUC rezultata. Slabiji rezultat moguće je uočiti pri usporedbi vrijednosti unakrsne provjere algoritma stabla odlučivanja s rezultatom u istraživačkim radom [2], međutim razlog tomu jest što je rezultat dobiven algoritmom stabla odlučivanja sklon osciliranju budući da je jedan od njegovih hiperparametara postavljen na opciju *random*.

8. ZAKLJUČAK

U ovom diplomskom radu analiziran je primjer jednostavnog skupa podataka koji je poslužio za treniranje i testiranje nekih od najpoznatijih algoritama strojnog učenja. Skup podataka odnosi se na vrednovanje automobila na osnovu njihovih glavnih karakteristika. Radi uvida u strukturu skupa podataka i saznanja kako pojedine karakteristike utječu na evaluaciju automobila odrađena je statistička analiza. Statističkom analizom ustanovilo se kako je riječ o klasifikacijskom skupu podataka čija je struktura zatim pojednostavljena zahvaljujući rezultatima korelacijske analize i statističkog testa kojima se dokazalo da broj vrata kod automobila nema značajan utjecaj prilikom evaluacije. Također, ustvrdilo se kako je riječ o neuravnoteženom skupu podataka, što je dalo na znanje koji od rezultata korištenih metrika su od većeg značaja prilikom ispitivanja kvalitete pojedinog algoritma, tj. modela. Algoritmi koji svoj rad temelje na različitim pretpostavkama 80% skupa podataka iskoristili su za učenje, a preostalih 20% za testiranje, nakon čega su kao konačnu izlaznu vrijednost ispisivali prosjek rezultata testiranja na podacima za učenje i za testiranje te na taj način omogućili otkrivanje eventualne pojave prekomjernog opremenja. Svaki algoritam je iterativnim postupkom nasumično odabirao različite vrijednosti hiperparametara s ciljem ostvarenja što boljeg rezultata na izlazu. S obzirom na relativno malu veličinu korištenog skupa podataka, defaultnim vrijednostima hiperparametara ostvarili su se relativno dobri rezultati zbog čega nije ostalo puno prostora za poboljšanje. Općenito, za slučaj ovog skupa podataka linearni modeli (logistička regresija i SGD) pokazali su se kao najlošiji odabir, dok su najbolje rezultate postigli MLP klasifikator te klasifikator stabla odlučivanja koje se pokazalo puno učinkovitijim od MLP-a zbog drastično manjeg vremena izvođenja. U zadnjem dijelu rada izvršena je metoda k-struke unakrsne validacije na temelju koje su se dobile stabilnije procjene izvedbe modela te se zaključilo da u ovom slučaju povećanje broja preklopa (koeficijenta k) neće izazvati značajno poboljšanje rezultata. Uz to, ustanovljeno je kako Stacking ansambl kreiran preko algoritama koji su postigli najbolje rezultate pri unakrsnoj provjeri nije uspio ostvariti bolji rezultat u odnosu na MLP i model stabla odlučivanja, što znači da je za eventualno poboljšanje rezultata potrebno odabrati bolje algoritme.

SAŽETAK

U ovom diplomskom radu analiziran je skup podataka evaluacije automobila koji je poslužio za treniranje i testiranje popularnih algoritama strojnog učenja, među koje spadaju logistička regresija, algoritam k-najbližih susjeda, višeslojni perceptron i stablo odlučivanja. Skup podataka je klasifikacijskog tipa te automobile vrednuje na osnovu njihovih karakteristika, kao što su cijena, trošak održavanja, razina sigurnosti i slično. Utjecaj vrijednosti pojedine karakteristike, koja odgovara ulaznoj varijabli, na procjenu automobila promatra se na osnovu grafičkih prikaza, hi-kvadrat testa i korelacijske analize. Na temelju donesenih zaključaka iz statističke analize započinje učenje i testiranje klasifikacijskih algoritama. Iterativnim postupkom nastoji se od svakog algoritma dobiti najbolje moguće rješenje, pritom se njihova kvaliteta ispituje na osnovu rezultata pojedinih metrika (točnost, preciznost, ROC AUC rezultat,...). Postizanje stabilne procjene izvedbe pojedinog modela te prikaz na koji način se modeli mogu kombinirati ostvareno je primjenom metoda unakrsne validacije i ansambla. Za potrebe izvođenja rada koristi se Spyder softver dizajniran za Python programski jezik.

Ključne riječi: strojno učenje, skup podataka, statistička analiza, hi-kvadrat test, Python, algoritmi klasifikacije, metrike, unakrsna validacija, metoda ansambla.

SUMMARY

In this thesis, a car evaluation data set was analized and used for training and testing popular machine learning algorithms, such as Logistic regression, k-nearest neighbor algorithm, Multi-Layer Perceptron and Decision Tree. The data set corresponds to classification type problem and evaluates cars based on their characteristics, such as price, maintenance cost, safety level, etc. Influence of the value of particural characteristic, which corresponds to the input variable, on evaluation of the car is observed on the basis of graphical representations, chi-square test and correlation analysis. Based on the conclusions reached from the statistical analysis begins learning and testing of classification algorithms. Each algorithm tries to achieve the best possible solution through an iterative process, while their quality is tested based on the results of particular metrics (accuracy, precision, ROC AUC score,...). Achieving stable performance evaluation of particular model and showing how models can be combined was described by applying the cross-validation and ensemble methods. To create this work is used Spyder software designed for Python programming language.

Keywords: machine learning, dataset, statistical analysis, chi-square test, Python, classification algorithms, metrics, cross-validation, ensemble method.

9. LITERATURA

[1] Rehman, Z. U., Fayyaz, H., Shah, A. A., Aslam, N., Hanif, M., & Abbas, S. (2018). Performance evaluation of MLPNN and NB: a comparative study on Car Evaluation Dataset. International Journal of Computer Science and Network Security, 18(9), 144-147.

[2] Awwalu, J., Ghazvini, A., & Bakar, A. A. (2014). Performance comparison of data mining algorithms: a case study on car evaluation dataset. Int. J. Comput. Trends Technol, 13(2), 78-82.

[3] S interneta: https://zaf.biol.pmf.unizg.hr/behaviour/Hi%20kvadrat%20test.pdf

[4] S interneta: https://passel2.unl.edu/view/lesson/9beaa382bf7e/8

[5] S interneta: https://study.com/learn/lesson/chi-square-distribution-graph-examples.html

[6] Vakili, M., Ghamsari, M. & Rezaei, M. (2020.). "Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification"

[7] S interneta: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

[8] Juba, B. & Le, H. S. (2019.). "Precision-Recall versus Accuracy and the Role of Large Data Sets". Washington University in St. Louis.

[9] Car, Z. (2022.). Predavanje: "Funkcije gubitaka i optimizacijski algoritmi". Sveučilište u Rijeci, Tehnički fakultet.

[10] Ying, X. (2019.). " An Overview of Overfitting and its Solutions ".

[11] Gupta, S. & Gupta, A. (2019.). "Dealing with Noise Problem in Machine Learning Data-sets: A Systematic Review". Manipal University Jaipur, India.

[12] S interneta: https://www.ibm.com/cloud/learn/overfitting

[13] Šnajder, J. (2020/2021.). "Strojno učenje: 2. Osnovni koncepti". Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva.

[14] Jurafsky, D. & H. Martin, J. (2020.). "Speech and Language Processing". Stanford University & University of Colorado at Boulder.

[15] S interneta: https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/

[16] S interneta: https://jermwatt.github.io/machine_learning_refined/notes/4_Second_order_methods/4_4_Newtons.html

[17] S interneta: https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/

[18] Šnajder, J. (2020/2021.). "Strojno učenje: 6. Logistička regresija". Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva.

[19] Durga Bhavani, D., Vasavi, A. Keshava, P.T. (2016). "Machine Learning: A Critical Review of Classification Techniques". Dept of CSE, NMREC, Hyderabad, India.

[20] S interneta: https://www.jcchouinard.com/k-nearest-neighbors/

[21] Cisłak. A. & Grabowski, S. (2014.). "Experimental evaluation of selected tree structures for exact and approximate k-nearest neighbor classification". Technical University of Munich & Lodz University of Technology.

[22] F. Evangelista, P., J. Embrechts, M., K. Szymanski, B. (2004.). "Taming the curse of dimensionality in kernels and novelty detection".

[23] S interneta: https://www.researchgate.net/figure/Biological-neural-network-2_fig8_267381125

[24] Zekić Sušac, M. (2021.). Predavanje:"Neuronske mreže". Sveučilište u Osijeku, Ekonomski fakultet u Osijeku.

[25] Car, Z., Anđelić, N., Lorencin, I. (2022.). Predavanje: "Uvod u umjetne neuronske mreže". Sveučilište u Rijeci, Tehnički fakultet.

[26] Car, Z. (2022.). Predavanje: "Umjetne neuronske mreže". Sveučilište u Rijeci, Tehnički fakultet.

[27] S interneta: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[28] Rokach, L. & Maimon, O. (2007.). "Data mining with decision trees". Ben-Gurion University & Tel-Aviv University.

[29] S interneta: https://elf11.github.io/2018/07/01/python-decision-trees-acm.html

[30] Steinbach, T. (2004.). "Classification: Basic Concepts, Decision Trees, and Model Evaluation".

[31] S interneta: https://scikit-learn.org/stable/modules/cross_validation.html

[32] García, V., Sánchez, J. S., & Marqués, A. I. (2019). Synergetic application of multi-criteria decision-making models to credit granting decision problems. Applied Sciences, 9(23), 5052.

[33] Maleki, F., Muthukrishnan, N., Ovens, K., Reinhold, C., Forghani, R. (2020.). "Machine Learning Algorithm Validation: From Essentials to Advanced Applications and Implications for Regulatory Certification and Deployment".

[34] J. Jain, B. (2016.). "Condorcet's Jury Theorem for Consensus Clustering and its Implications for Diversity". Technical University of Berlin, Germany.

[35] Pintelas, P. & E. Livieris, I. (2020.). "Ensemble Algorithms and Their Applications". University of Patras, Greece.

[36] T. Ngo, K. (2018.). "Stacking Ensemble for auto ml". Faculty of the Virginia Polytechnic Institute.

[37] S interneta: https://www.analyticsvidhya.com/blog/2021/08/ensemble-stacking-for-machine-learning-and-deep-learning/

[38] Ekman, D. & Härner, S. (2022.). "Comparing Ensemble Methods with Individual Classifiers in Machine Learning for Diabetes Detection".

10. POPIS SLIKA

11. POPIS TABLICA

12. DODATAK A – Statistička analiza

Učitavanje skupa podataka i vizualizacija raspodjele vrijednosti varijabli:

```
"""
@author: Luka Ilijanic
"""

import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

file _name = "C:/Users/car_evaluation.csv"
variable_names = ["buying_price", "maint_cost", "no_doors", "no_persons", "lug_boot", "safety", "decision"]
data = pd.read_csv(file_name, header = None, names = variable_names)
print(data)
print(data.dtypes)

def bar_graph(data1, column_name):
    table = pd.crosstab(index = data1[column_name], columns="count")
    table.plot.bar()
    plt.xticks(rotation=0)
    plt.legend()
    plt.rcParams['axes.spines.left'] = False
    plt.rcParams['axes.spines.right'] = False
    plt.rcParams['axes.spines.top'] = False
    plt.rcParams['axes.spines.bottom'] = True
    plt.grid(axis = 'y')
    plt.rc('axes', axisbelow=True)
    return table

table_decision = bar_graph(data,"decision")
print(table_decision)
table_buying_price = bar_graph(data,"buying_price")
print(table_buying_price)
table_maint_cost = bar_graph(data,"maint_cost")
print(table_maint_cost)
table_no_doors = bar_graph(data,"no_doors")
print(table_no_doors)
table_no_persons = bar_graph(data,"no_persons")
print(table_no_persons)
table_lug_boot = bar_graph(data,"lug_boot")
print(table_lug_boot)
table_safety = bar_graph(data,"safety")
print(table_safety)
```

Funkcija `crosstab` služi za izračunavanje jednostavne unakrsne tablice s dva ili više faktora, a naredbom `plot.bar()` dobivena tablica pretvara se u stupčasti graf koji prikazuje kategoričke podatke pravokutnim trakama s duljinama proporcionalnim vrijednostima koje predstavljaju.

Vizualizacija uparenih stupčastih dijagrama:

```
def paired_bar_graph(data1,column_name1,column_name2):
    table = pd.crosstab(index = data1[column_name1], columns = data1[column_name2])
    table.plot.bar()
    plt.xticks(rotation=0)
    plt.rcParams['axes.spines.left'] = False
    plt.rcParams['axes.spines.right'] = False
    plt.rcParams['axes.spines.top'] = False
    plt.rcParams['axes.spines.bottom'] = True
    plt.grid(axis = 'y')
    plt.rc('axes', axisbelow=True)
    plt.legend(bbox_to_anchor=(1.04, 0.7), loc="upper left")
    return table

table_bprice_decision = paired_bar_graph(data,"buying_price","decision")
print(table_bprice_decision)
table_mcost_decision = paired_bar_graph(data,"maint_cost","decision")
print(table_mcost_decision)
table_nodoors_decision = paired_bar_graph(data,"no_doors","decision")
print(table_nodoors_decision)
table_nopersons_decision = paired_bar_graph(data,"no_persons","decision")
print(table_nopersons_decision)
table_lboot_decision = paired_bar_graph(data,"lug_boot","decision")
print(table_lboot_decision)
table_safety_decision = paired_bar_graph(data,"safety","decision")
print(table_safety_decision)
```

Mapiranje i izrada korelacijske tablice:

```
map_decision = {'unacc':0,'acc':1,'good':2,'vgood':3}
map_bprice = {'low':0, 'med':1, 'high':2, 'vhigh':3}
map_mcost = {'low':0, 'med':1, 'high':2, 'vhigh':3}
map_nodoors = {'2':0, '3':1, '4':2, '5more':3}
map_nopersons = {'2':0, '4':1, 'more':2}
map_lugboot = {'small':0, 'med':1, 'big':2}
map_safety = {'low':0, 'med':1, 'high':2}

coded_data = pd.DataFrame()
coded_data['buying_price'] = data['buying_price'].map(map_bprice)
coded_data['maint_cost'] = data['maint_cost'].map(map_mcost)
coded_data['no_doors'] = data['no_doors'].map(map_nodoors)
coded_data['no_persons'] = data['no_persons'].map(map_nopersons)
coded_data['lug_boot'] = data['lug_boot'].map(map_lugboot)
coded_data['safety'] = data['safety'].map(map_safety)
coded_data['decision'] = data['decision'].map(map_decision)

corr = coded_data.corr()
print(corr)
plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, linewidth=0.5)
plt.xticks(rotation=30)
```

Korištenjem funkcije `corr` izvodi se korelacija tablice ordinalnih varijabli. Radi jasnije vizualizacije primjenjuje se funkcija `heatmap` iz knjižnice *Seaborn*.