Estimacija energetske učinkovitosti stambenih zgrada primjenom strojnog učenja

Vukić, Tomislav

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

ESTIMACIJA ENERGETSKE UČINKOVITOSTI STAMBENIH ZGRADA PRIMJENOM STROJNOG UČENJA

Mentor: Prof. dr. sc. Zlatan Car

Komentor: V. asist. dr. sc. Nikola Anđelić

Rijeka, studeni 2023.

Tomislav Vukić
0069075038

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 14. srpnja 2023.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Evolucijska robotika**
Grana: **2.03.06 automatizacija i robotika**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Tomislav Vukić (0069075038)**
Studij: Sveučilišni diplomski studij elektrotehnike
Modul: Automatika

Zadatak: **Estimacija energetske učinkovitosti stambenih zgrada primjenom strojnog učenja**

Opis zadatka:

Napraviti pregled postojećih istraživanja estimacije energetske učinkovitosti zgrada primjenom različitih metoda strojnog učenja. Na javno dostupnom skupu podataka napraviti statističku analizu i pred obradu podataka. Primijeniti različite metode strojnog učenja za estimaciju energetske učinkovitosti te ispitati utjecaj različitih metoda skaliranja i normalizacije podataka na točnost estimacije. Primijeniti unakrsnu validaciju za treniranje metoda strojnog učenja s nasumičnim pretraživanjem hiperparametara. Razviti ansambl metode strojnog učenja te ispitati mogućnost poboljšanja točnosti estimacije energetske učinkovitosti.

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

Zadatak uručen pristupniku: 14. srpnja 2023.

Mentor:

Prof. dr. sc. Zlatan Car

Nikola Anđelić

Dr. sc. Nikola Anđelić (komentor)

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Dubravko Franković

ESTIMACIJA ENERGETSKE UČINKOVITOSTI STAMBENIH ZGRADA PRIMJENOM STROJNOG UČENJA

Sukladno članku 8. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci od 31. siječnja 2020., izjavljujem da sam samostalno izradio/izradila diplomski rad prema zadatku preuzetom dana 14.07.2023.

Rijeka, rujan 2023.

_____

Tomislav Vukić

Želim bih iskoristiti ovu priliku da izrazim svoju duboku zahvalnost svima koji su mi pomogli tijekom procesa izrade ovog diplomskog rada.

Prvenstveno se želim zahvaliti svom mentoru prof. dr. sc. Caru i komentoru v. asist. dr. sc. Anđeliću na neizmjernom strpljenju, vodstvu i stručnosti koje ste mi pružili tijekom izrade ovog rada.

Također se želim zahvaliti svojim roditeljima koji su mi pružili neizmjeran moralni, emocionalni i financijski oslonac tijekom mog čitavog školovanja. Vaša podrška i vjera u mene su mi bili neizmjerno važni tijekom ovog puta.

Naposlijetku, zahvaljujem se svim prijateljima i članovima obitelji koji su bili uz mene.

SADRŽAJ

1. UVOD

1.1. Energetska učinkovitost

Energetska učinkovitost ili efikasnost može se definirati kao skup postupaka čija je svrha smanjenje energetske potrošnje uz postizanje zadovoljavajućih rezultata. Energetski učinkovite zgrade i proizvodni pogoni zahtijevaju manje energije za grijanje i hlađenje, te za pokretanje električnih uređaja.

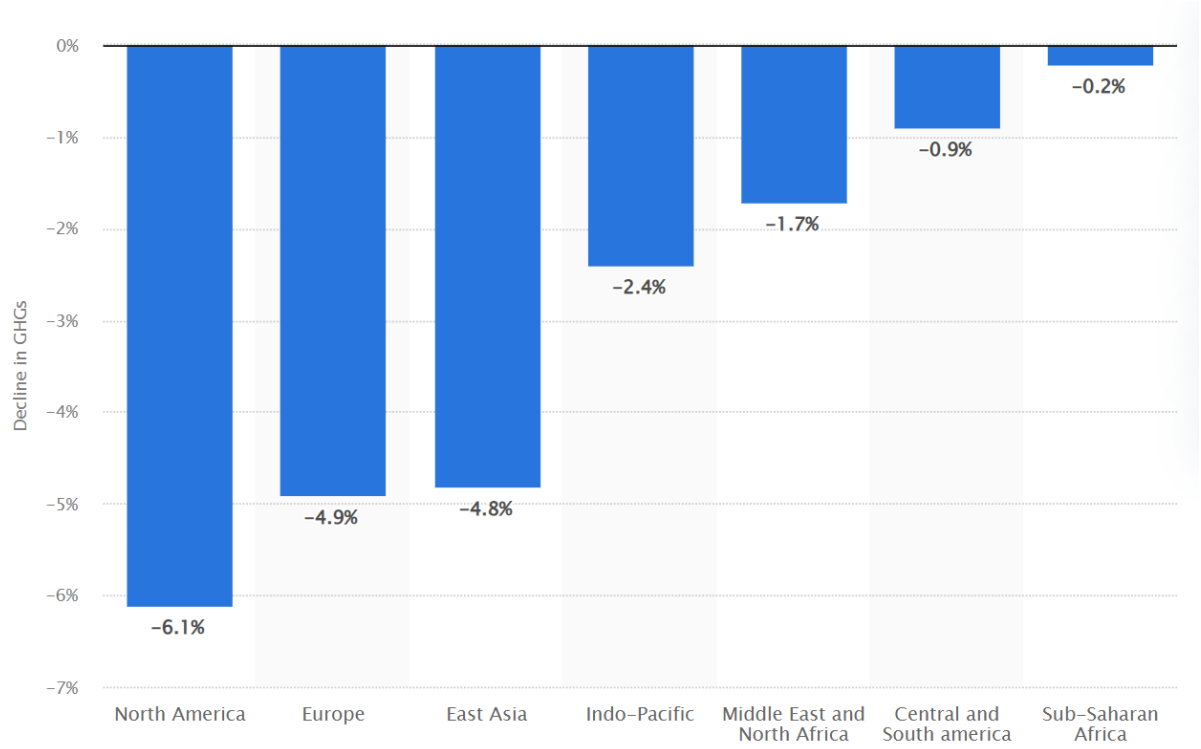
Primarne beneficije povećanja učinkovitosti energetske potrošnje su ekonomske i ekološke prirode. Osim što pružaju mogućnost smanjenja režija za pojedince i poslovna poduzeća razvitak energetski učinkovitih tehnologija izravno doprinosi smanjenju udjela ugljikovog dioksida u atmosferi i pomaže u borbi protiv klimatskih promjena.

Budući da nam daje mogućnost analize i obrade velikih količina podataka relativno brzo, danas ključnu ulogu u poboljšanju energetske efikasnosti ima umjetna inteligencija.

1.2. Pregled postojećih istraživanja

Umjetna inteligencija smatra se generalnom tehnologijom koja potiče druge inovacije [1]. Ona ubrzava razvoj novih tehnologija čije se prednosti manifestiraju kroz uštedu energije i čišćoj proizvodnji što direktno poboljšava energetsku efikasnost proizvodnih poduzeća [2].

Istraživanje [3], koje je napravljeno na 400 različitih organizacija iz područja automobilske industrije, industrijske proizvodnje, energetike i komunalnih usluga, pokazalo je da gotovo polovica (48%) koristi umjetnu inteligenciju za djelovanje na klimatske promjene. To je rezultiralo smanjenjem emisija stakleničkih plinova za 12.9%, poboljšanjem energetske učinkovitosti za 10.9%, te smanjenjem otpada za 11.7% u razdoblju od 2017. godine do 2020. godine.



Slika 1.1. Utjecaj AI tehnologija na emisije stakleničkih plinova [4]

Slika 1.1. prikazuje procjene utjecaja implementacije AI tehnologija na smanjenje emisija stakleničkih plinova za 2030. godinu po različitim regijama. Za Sjevernu Ameriku predviđa se smanjenje emisija za 6.1%, za Europu 4.9%, za zemlje istočne Azije 4.8%, za Indo-pacifičku regiju 2.4%, a za sve ostale regije očekuje se stopa smanjenja emisija manja od 2%.

Zbog rastućih briga oko rasipanja energije i utjecaja na okoliš u prethodna dva desetljeća napisan je velik broj radova koji istražuju energetske performanse zgrada. Istraživanja pokazuju zabrinjavajući porast potrošnje energije u zgradama, te da su grijanje, ventilacija i hlađenje vodeći čimbenici tog porasta [5]. Jedan od načina za ublažavanje sve veće potražnje za dodatnom opskrbom energije je energetski učinkovitiji dizajn zgrada s poboljšanim svojstvima očuvanja energije.

Korištenje softvera za simulaciju energetske potrošnje zgrada je u širokoj upotrebi danas, međutim takav pristup zahtjeva puno vremena i stručno poznavanje korištenih softvera. Iz tog razloga se brojni istraživači služe tehnikama strojnog učenja za proučavanje učinka različitih parametara zgrade. Takav pristup je lakši i brži od korištenja softvera za simulaciju.

Brojne tehnike umjetne inteligencije već su korištene u kontekstu energetske učinkovitosti zgrada , npr. algoritam potpornih vektora [6], umjetne neuronske mreže [7], itd. Alati strojnog učenja također su eksplicitno korišteni i za predviđanje toplinskog dobitka i opterećenja prostora [8]. U istraživanju [9] utvrđeno je da rezultati dobivenim tehnikama umjetne inteligencije odstupaju od rezultata dobivenih korištenjem naprednih simulacijskih softvera odstupaju za maksimalno 5%.

1.3. Hipoteza istraživanja

Za učinkovito projektiranje zgrade neophodno je provesti izračune za određivanje toplinskog dobitka i opterećenja prostora u svrhu pravilnog odabira specifikacije opreme za grijanje i hlađenje s ciljem održavanja ugodne temperature unutar zgrade. Glavna hipoteza ovog rada je mogućnost implementacije tehnika strojnog učenja za te potrebe i razvoj dovoljno preciznog modela za predviđanje tih dviju varijabli. Kako bi se ta hipoteza ispitala poduzeti će se sljedeći koraci:

- Istražiti kakvi se rezultati dobivaju koristeći zadane hiperparametre,
- Istražiti promjene u rezultatima nakon skaliranja i normalizacije skupa podataka,
- Istražiti postoje li poboljšanja u rezultatima nakon nasumičnog pretraživanja hiperparametara,
- Istražiti postoji li opasnost od pretreniranja,
- Implementirati određene ansambl tehnike kako bi se dobio robustan model za predviđanje.

1.4. Pregled rada

Kako bi se riješili prethodno spomenuti izazovi struktura ovog diplomskog rada formirana je na sljedeći način:

- Poglavlje 2 – Opis i analiza skupa podataka – ovo poglavlje daje kratki opis skupa podataka koji je korišten u istraživanju, prikazuje rezultate statističke analize i sadrži objašnjenje i dobivene rezultate korelacijske analize,
- Poglavlje 3 – Korišteni algoritmi i evaluacijske metrike – daje kratki pregled 9 algoritama koji su korišteni za razvoj modela za predikciju i objašnjenje evaluacijskih metrika kojima se određuje kvaliteta razvijenog modela,
- Poglavlje 4 – Metode skaliranja i normalizacije – u poglavlju se opisuje 5 različitih metoda skaliranja i normalizacije koje su primijenjene u ovom radu,
- Poglavlje 5 – Inicijalna analiza i rezultati – poglavlje sadrži rezultate modela koji su dobiveni bez skaliranja i normalizacije skupa podataka za zadane hiperparametre,
- Poglavlje 6 – Analiza i rezultati dobiveni metodama skaliranja i normaliziranja – poglavlje sadrži rezultate modela koji su dobiveni nakon primjene 5 metoda skaliranja i normalizacije,
- Poglavlje 7 – Nasumični odabir hiperparametara – poglavlje daje opis postupka nasumičnog pretraživanja hiperparametara, sadrži rezultate modela i hiperparametre koji su dali te rezultate za modele razvijene bez i sa implementacije metoda skaliranja i normalizacije,
- Poglavlje 8 – Unakrsna validacija – poglavlje sadrži opis pojave pretreniranja i postupka unakrsne validacije, sadrži rezultate modela i hiperparametre koji su dali te rezultate za modele razvijene bez i sa implementacije metoda skaliranja i normalizacije,
- Poglavlje 9 – Primjena ansambl metoda – poglavlje daje opis ansambl tehnika koje su korištene za potrebe ovog rada, sadrži rezultate modela i hiperparametre koji su dali te rezultate za modele razvijene bez i sa implementacije metoda skaliranja i normalizacije,
- Poglavlje 10 – Zaključak – u ovom poglavlju dan je kratki pregled razloga za razvijanje modela za predikciju energetske efikasnosti zgrada, opisan je postupak izrade ovog rada, te njegov finalni zaključak.

2. OPIS I ANALIZA SKUPA PODATAKA

Skup podataka koji je korišten u radu proizašao je iz analize 12 različitih oblika zgrada simuliranih u programskom paketu *Autodesk Ecotect Analysis*.

Korišteni skup podataka sadrži 8 značajki po kojima se zgrade razlikuju te 2 dobivena odziva. Cijeli skup podataka sastoji se od 768 uzoraka. Značajke su redom definirane kao: relativna kompaktnost (X1), površina poda (X2), površina zida (X3), površina krova (X4), visina (X5), orijentacija (X6), površina prozora (X7) i raspodjela površine prozora (X8). Dobiveni odzivi su toplinski dobitak prostora (Y1) i toplinsko opterećenje (Y2). Cilj analize je bio da se korištenjem 8 značajki predvide oba odziva.

2.1. Statistička analiza

Na skupu podataka provedena je statistička analiza i za svaku varijablu su dobivene minimalna i maksimalna vrijednost, standardna devijacija, srednja vrijednost, te donji, gornji i srednji kvartili (medijan).

Tablica 2.1. Statistička analiza

VARIJABLA	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
BROJ PODATAKA	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0	768.0
SREDNJA VRIJEDNOST	0.764	671.7	318.5	176.6	5.250	3.50	0.234	2.812	22.30	24.58
STANDARDNA DEVIJACIJA	0.106	88.08	43.62	45.16	1.751	1.118	0.133	1.550	10.09	9.513
MINIMALNA VRIJEDNOST	0.620	514.5	245.0	110.2	3.50	2.0	0.0	0.0	6.010	10.9
DONJI KVARTIL [25%]	0.683	606.3	294.0	140.8	3.50	2.750	0.1	1.750	12.99	15.62
SREDNJI KVARTIL [50%]	0.750	673.7	318.5	183.7	5.250	3.50	0.25	3.0	18.95	22.08

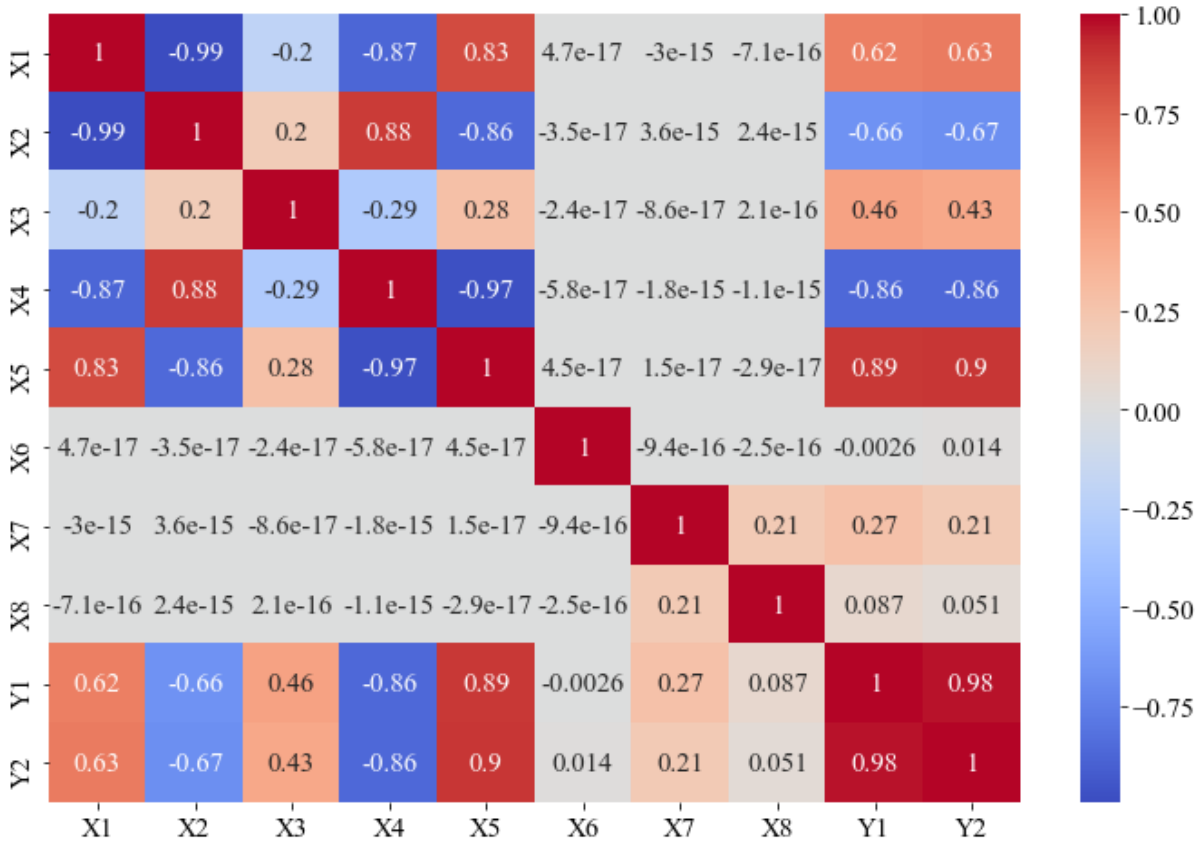
GORNJI KVARTIL [75%]	0.830	741.1	343.0	220.5	7.0	4.25	0.4	4.0	31.66	33.13
MAKSIMALNA VRIJEDNOST	0.980	808.5	416.5	220.5	7.0	5.0	0.4	5.0	43.10	48.03

2.2. Korelacijska analiza

Korelaciju definiramo kao mjeru povezanosti između dvije ili više varijabli. Povezanost između varijabli znači da možemo poznavanjem jedne varijable predvidjeti vrijednost druge. Tu povezanost izražavamo korištenjem Pearsonovog koeficijenta korelacije [10].

Pearsonov koeficijent korelacije poprima vrijednosti od -1 do 1 s tim da nam vrijednosti veće od nule znače pozitivnu, a vrijednosti manje od nule negativnu korelaciju. Vrijednost koeficijenta 1 označava maksimalnu pozitivnu korelaciju što bi značilo da linearni rast jedne varijable rezultira linearnim rastom druge. Vrijednost -1 označava maksimalnu negativnu korelaciju što znači da linearni porast jedne varijable rezultira linearnim padom druge. Ukoliko koeficijent korelacije poprima vrijednosti veće od +0.8 ili manje od -0.8 smatramo da postoji jaka korelacija između varijabli. Sukladno tome za vrijednosti manje od +0.8 ili veće od -0.8 kažemo da je korelacija između varijabli slaba. Kad Pearsonov koeficijent korelacije poprimi vrijednost 0 znamo da ne postoji međusobna povezanost varijabli [10].

Za dani skup podataka provedena je korelacijska analiza čiji su rezultati prikazani na sljedećoj slici.



Slika 2.1. Korelacijska mapa ulaznih i izlaznih podataka

Provedena korelacijska analiza pokazala je da najbolju korelaciju sa izlaznom varijablom Y1 imaju ulazni parametri X4 i X5, a najlošiju ulazni parametri X6 i X8. Nadalje sa izlaznom varijablom Y2 također najbolju korelaciju imaju ulazni parametri X4 i X5, a najlošiju parametri X6 i X8.

3. ALGORITMI STROJNOG UČENJA I EVALUACIJSKE METODE

Strojno učenje je grana umjetne inteligencije koja se fokusira na uporabu podataka i algoritama kako bi se imitirao način na koji ljudi uče [11]. Jedan od osnovnih ciljeva strojnog učenja je razvoj algoritama koji imaju sposobnost davanja što točnijih rezultata na nove podatke nakon treniranja na prethodnim podacima. Za te potrebe rada dani skup podataka je podijeljen u dvije skupine. Sedamdeset posto podataka nasumično je odabrano i korišteno za razvoj algoritma koji će imati mogućnost relativno točnog predviđanja dviju izlaznih varijabli, toplinskog dobitka prostora (Y1) i toplinskog opterećenja (Y2), a ostalih trideset posto podataka će biti iskorišteno za testiranje takvog algoritma.

Nakon razdvajanja skupa podataka na njima je potrebno primijeniti određene metode koje će istrenirati model, tj. provesti postupak kojim model uči vezu između ulaznih i izlaznih varijabli, a zatim ga i testirati. Za potrebe ovog rada upotrijebiti će se sljedećih 9 metoda: „LinearRegression“, „Ridge“, „SGDRegressor“, „Lars“, „Lasso“, „LassoLars“, „ARDRegression“, „BayesianRidge“ i „MLPRegressor“. U nastavku će se dati kratki opis svih navedenih metoda i navesti će se njihovi hiperparametri.

Metoda „LinearRegression“ je jedna od najvažnijih i najčešće korištenih regresijskih tehnika, a ujedno je i najjednostavnija regresijska metoda. Njena glavna prednost je jednostavnost kojom se mogu interpretirati rezultati. Metoda se, kako joj i ime nalaže, temelji na linearnoj regresiji. Cilj linearne regresije je prilagoditi liniju kroz zadani skup točaka, pritom birajući najbolje prilagođavanje [12].

Tablica 3.1. Hiperparametri „LinearRegression“ metode

Ime hiperparametra	**Tip podataka**
fit_intercept	Bool
normalize	Bool
copy_X	Bool
n_jobs	Integer
positive	Bool

Metoda „Ridge“ je tehnika regularizacije koja penalizira L2-normu koeficijenata u linearnoj regresiji. Jedan od izazova korištenja ove metode je potreba za postavljanjem hiperparametra (α) koji kontrolira količinu regularizacije [7].

Tablica 3.2. Hiperparametri „Ridge“ metode

Ime hiperparametra	Tip podataka	Ime hiperparametra	Tip podataka
alpha	Float	tol	Float
fit_intercept	Bool	solver	/
copy_X	Bool	random_state	Integer
max_iter	Integer	normalize	Bool

Hiperparametar „solver“ uključuje sljedeće:

- *auto* – automatski odabir „solver-a“ na temelju tipu danih podataka,
- *svd* – koristi dekompoziciju na singularne vrijednosti za izračunavanje Ridge koeficijenata,
- *cholesky* – koristi standardnu funkciju *„scipy.linalg.solve“* za dobivane rješenja zatvorenog oblika,
- *sparse_cg* – koristi alat za rješavanje konjugiranih gradijenata,
- *lsqr* – korisni regulariziranu rutinu najmanjih kvadrata,
- *sag* – koristi stohastički gradijentni pad, i
- *lbfgs* – koristi L-BFGS-B algoritam.

Metoda „SGDRegressor“ je algoritam strojnog učenja koji implementira stohastički gradijentni pad za rješavanje problema regresije. To je popularan izbor za regresijske zadatke velikih razmjera zbog njegove sposobnosti rukovanja visokodimenzionalnim skupovima podataka i kratkog vremena obuke [13].

Tablica 3.3. Hiperparametri „SGDRegressor“ metode

Ime hiperparametra	Tip podataka	Ime hiperparametra	Tip podataka
loss	String	epsilon	Float
penalty	/	random_state	Integer
alpha	Float	learning_rate	String
l1_ratio	Float	eta0	Float
fit_intercept	Bool	power_t	Float
max_iter	Integer	early_stopping	Bool
tol	Float	validation_fraction	Float
shuffle	Bool	n_iter_no_change	Integer
verbose	Integer	warm_start	Bool
average	Bool ili Integer		

Hiperparametar „penalty“ uključuje: *„l2“*, što je standardni regularizator za linearne SVM modele, te *„l1“* i *„elasticnet“* koji mogu unijeti rijetkost u model.

Metoda „Lars“ je algoritam koji se koristi u regresiji za visokodimenzionalne podatke (tj. podatke s velikim brojem atributa). Budući da se koristi za takve podatke, LARS u svakom koraku pronalazi atribut koji najviše korelira s ciljanom vrijednošću [14].

Tablica 3.4. Hiperparametri „Lars“ metode

Ime hiperparametra	**Tip podataka**	**Ime hiperparametra**	**Tip podataka**
fit_intercept	Bool	eps	Float
verbose	Bool ili Integer	copy_X	Bool
normalize	Bool	fit_path	Bool
nrecompute	Bool	jitter	Float
n_nonzero_coefs	Integer	random_state	Integer

Metoda „Lasso“ (*Least absolute shrinkage and selection operator*) se koristi i za regularizaciju i za odabir modela. Ako model koristi L1 tehniku regularizacije, onda kažemo da se to zove Lasso regresija [15].

Tablica 3.5. Hiperparametri „Lasso“ metode

Ime hieperparametra	**Tip podataka**	**Ime hiperparametra**	**Tip podataka**
alpha	Float	tol	Float
fit_intercept	Bool	warm_start	Bool
precompute	Bool	positive	Bool
copy_X	Bool	random_state	Integer
max_iter	Integer	selection	/

Hiperparametar „selection“ može poprimiti vrijednosti „random“ i „cyclic“. Odabirom vrijednosti „random“ se nasumični koeficijent ažurira prilikom svake iteracije.

Kod metode „LassoLars“ Lasso model je je treniran sa regresijom najmanjeg kuta. To je linearni model koji je prethodno treniran s L1 kao regulatorom [16].

Tablica 3.6. Hiperparametri „LassoLars“ metode

Ime hiperparametra	**Tip podataka**	**Ime hiperparametra**	**Tip podataka**
alpha	Float	eps	Float
fit_intercept	Bool	copy_X	Bool
verbose	Bool ili Integer	fit_path	Bool
normalize	Bool	positive	Bool
precompute	Bool	jitter	Float
max_iter	Integer	random_state	Integer

„ARDRegression“ metoda se bazira na Bayesovoj metodi zaključivanja. Ova metoda promatra težine modela kao da su raspodijeljeni po Gaussovoj krivulji i procjenjuje hiperparametre α i λ kroz iteraciju [17].

Tablica 3.7. Hiperparametri „ARDRegression“ metode

Ime hiperparametra	**Tip podataka**	**Ime hiperparametra**	**Tip podataka**
n_iter	Integer	compute_score	Bool
tol	Float	treshold_lambda	Float
alpha_1	Float	fit_intercept	Bool
alpha_2	Float	copy_X	Bool
lambda_1	Float	verbose	Bool
lambda_2	Float	normalize	Bool

Bayesova regresija omogućuje prirodnom mehanizmu da preživi nedovoljno podataka ili loše distribuirane podatke formuliranjem linearne regresije korištenjem distributera vjerojatnosti umjesto točkastih procjena. Pretpostavlja se da je rezultat ili izlaz izvučen iz distribucije vjerojatnosti, a ne procijenjen kao jedna vrijednost. Jedna od najkorisnijih vrsta Bayesove regeresije je „BayesianRidge“ metoda koja procjenjuje probabilistički model problema regresije [18].

Tablica 3.8. Hiperparametri „BayesianRidge“ metode

Ime hiperparametra	**Tip podataka**	**Ime hiperparametra**	**Tip podataka**
n_iter	Integer	alpha_init	Float
tol	Float	lambda_init	Float
alpha_1	Float	compute_score	Bool
alpha_2	Float	fit_intercept	Bool
lambda_1	Float	copy_X	Bool
lambda_2	Float	verbose	Bool
normalize	Bool		

„MLPRegressor“ [19] je model umjetne neuronske mreže koji koristi povratno širenje za podešavanje težine između neurona kako bi se poboljšala točnost previđanja. Metoda implementira algoritam višeslojnog perceptrona za obuku i testiranje skupova podataka. Uključuje nekoliko hiperparametara koji se mogu koristiti za fino podešavanje performansi modela, uključujući broj skrivenih slojeva, aktivacijske funkcije, itd.

Tablica 3.9. Hiperparametri „MLPRegressor“ metode

Ime hiperparametra	Tip podataka	Ime hiperparametra	Tip podataka
hidden_layer_sizes	Array	tol	Float
activation	/	verbose	Bool
solver	/	warm_start	Bool
alpha	Float	momentum	Float
batch_size	Integer	nesterovs_momentum	Bool
learning_rate	/	early_stopping	Bool
learning_rate_init	Float	validation_fraction	Float
power_t	Float	beta_1	Float
max_iter	Integer	beta_2	Float
shuffle	Bool	epsilon	Float
n_iter_no_change	Integer	max_fun	Integer
random_state	Integer		

Hiperparametar „activation“ uključuje sljedeće:

- *identity* – aktivacijska funkcija identiteta,
- *logistic* – logistička sigmoidna funkcija,
- *tanh* – tangens hiperbolični, i
- *relu* – ispravljena funkcija linearne jedinice.

Hiperparametar „solver“ uključuje sljedeće:

- *lbfgs* – je optimizator iz obitelji kvazi-Newtonovih metoda,
- *sgd* – je stohastički gradijentni pad, i
- *adam* – je tip optimizatora koji se temelji na stohastičkom gradijentu.

Hiperparametar „learning_rate“ uključuje sljedeće:

- *constant* – konstantno brzina učenja,
- *invscaling* – postupno smanjenje brzine učenja, i
- *adaptive* – održava brzinu učenja konstantnom dokle god se gubitak treniranja nastavlja smanjivati.

Evaluacijske metode koje će se koristiti u ovom radu su: kvadratna greška, srednja apsolutna greška i korijen srednje kvadratne greške.

Kvadratna greška (R^2), ili koeficijent determinacije, se definira kao udio varijacije ovisne varijable koji se objašnjava regresijskom jednadžbom nezavisne varijable. Računa se dijeljenjem sume kvadrata greške procjene sa sumom kvadrata u kojem je od stvarne vrijednosti oduzeta srednja vrijednost. Rezultati dobiveni ovom metodom poprimaju vrijednosti od 0 do 1, a što je dobiveni rezultat veći zaključujemo da postoji bolja poveznica između procijenjene i stvarne vrijednosti [20],

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2}. \tag{3.1}$$

Srednja apsolutna greška (Mean absolute error, MAE) računa se kao suma apsolutnih vrijednosti razlika stvarnih i procijenjenih vrijednosti podijeljena sa brojem podataka [21]. Što je srednja apsolutna greška manja to je bolja povezanost između procijenjene i stvarne vrijednosti,

$$MAE = \frac{1}{N} * \sum_{i=1}^{N} |y_i - \hat{y}_i|. \tag{3.2}$$

Korijen srednje kvadratne greške (Root mean squared error, RMSE) je korijen prosječne kvadratne razlike između stvarnih i procijenjenih vrijednosti. Računa se korjenovanjem sume kvadrata greške procijene (koja se računa oduzimanjem stvarnih vrijednosti i procijenjenih vrijednosti) podijeljene sa brojem podataka [22],

$$RMSE = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}. \tag{3.3}$$

Kvaliteta izgrađenog modela ocjenjivati će se na način da će se odrediti srednja vrijednost i standardna devijacija rezultata prethodno objašnjenih evaluacijskih metrika dobivenih na podacima za treniranje i podacima za testiranje. Te vrijednosti usporediti će se sa vrijednostima koje su dali ostali modeli kako bi se odredio koji je najkvalitetniji [23]. U tu svrhu upotrijebiti će se sljedeće formule:

$$\overline{R^2} = \frac{R^2_{train} + R^2_{test}}{2}, \tag{3.4}$$

$$\overline{MAE} = \frac{MAE_{train} + MAE_{test}}{2}, \tag{3.5}$$

$$\overline{RMSE} = \frac{RMSE_{train} + RMSE_{test}}{2}, \tag{3.6}$$

$$R^2_{STD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(R_i^2 - \overline{R^2}\right)^2}, \tag{3.7}$$

$$MAE_{STD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(MAE_i - \overline{MAE})^2}, \tag{3.8}$$

$$RMSE_{STD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(RMSE_i - \overline{RMSE})^2}. \tag{3.9}$$

U svrhu jednostavnije interpretacije rezultati dobiveni prethodnim formulama će se grafički prikazati.

4. METODE SKALIRANJA I NORMALIZACIJE

Pretprocesiranje podataka je bitan korak za postizanje dobre performanse klasifikacije prije evaluacije podataka na algoritmima strojnog učenja. Neobrađeni skup podataka često može sadržavati značajke različitih razmjera i amplitudnih varijacija. Ova raznolikost može dovesti do problema u treniranju modela. Uzmimo za primjer skup podataka koji se koristi za medicinsko istraživanje. U tom slučaju možemo imati značajke kao što su visina i dob. Obje te značajke mogu imati potpuno različite opsege vrijednosti. Visina može varirati recimo između 1.50 i 2.00 metra, dok dob može biti između 0 i 100 godina. Bez pretprocesiranja modeli strojnog učenja mogu interpretirati značajke s većim vrijednostima kao važnije, što može kao posljedicu uzorkovati to da se istrenira model sa lošim performansama [25].

Pretprocesiranje uključuje različite metode, među kojima su i postupci skaliranja i normalizacije. Skaliranje i normalizacija su postupci pretprocesiranja skupa podataka kod kojih se mijenjaju vrijednosti atributa prema zajedničkoj skali ili rasponu u svrhu poboljšavanja performansi algoritma strojnog učenja [26].

Metode koje su korištene u svrhu pretprocesiranja podataka u ovom radu su redom: „StandardScaling“, „MinMaxScaler“, MaxAbsScaler“, „Normalizer“ i „PowerTransformer“.

Skaliranje možemo definirati kao tehniku za transformaciju skupa podataka na način da sve vrijednosti padaju unutar određenog raspona, primjerice od 0 do 1 ili od 0 do 100. Skaliranje se može primijeniti na različite načine, ali najosnovniji pristup je korištenjem „StandardScaling“ metode [27]. Kod ove metode podaci se skaliraju oduzimanjem srednje vrijednosti od svih podatkovnih točaka i dijeljenjem dobivenih vrijednosti sa standardnom devijacijom podataka [28]. Formula ove metode dana je u nastavku:

$$x_{new} = \frac{x_{old} - u}{s}, \tag{4.1}$$

gdje je *u* srednja vrijednost značajke i *s* standardna devijacija značajke.

Još jedna verzija standardnog skaliranja je „MinMaxScaler“ metoda. Ona nam omogućava skaliranje vrijednosti između određenih granica (uglavnom između 0 i 1) [29]. Postupak ovog tipa skaliranja se provodi na način da se prvo za svaki atribut u skupu podataka pronađu njegova minimalna i maksimalna vrijednost. Zatim se za svaku vrijednost u atributu izračunava nova

skalirana vrijednost na način da se od atributa oduzima njegova minimalna vrijednost, a zatim se to dijeli sa razlikom njegove maksimalne i minimalne vrijednosti [30]. To se postiže sljedećom formulom:

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}. \tag{4.2}$$

Ova metoda ne mijenja raspored ili raspodjelu podataka, već ih samo skalira.

Metoda „MaxAbsScaler“ je također često korištena tehnika skaliranja kod koje se razlika između podatkovnih točaka i minimalne vrijednosti dijeli sa maksimalnom vrijednošću [30]. Razlikuje se od standardnih alata za skaliranje jer skalira vrijednost svih podataka tako da padaju između vrijednosti -1 i 1 [29]. Kod ove metode prvo se pronalazi maksimalna apsolutna vrijednost za svaki atribut, a zatim se svaka vrijednost atributa skalira prema sljedećoj formuli:

$$x_{new} = \frac{x_{old}}{max_{abs}(x)}. \tag{4.3}$$

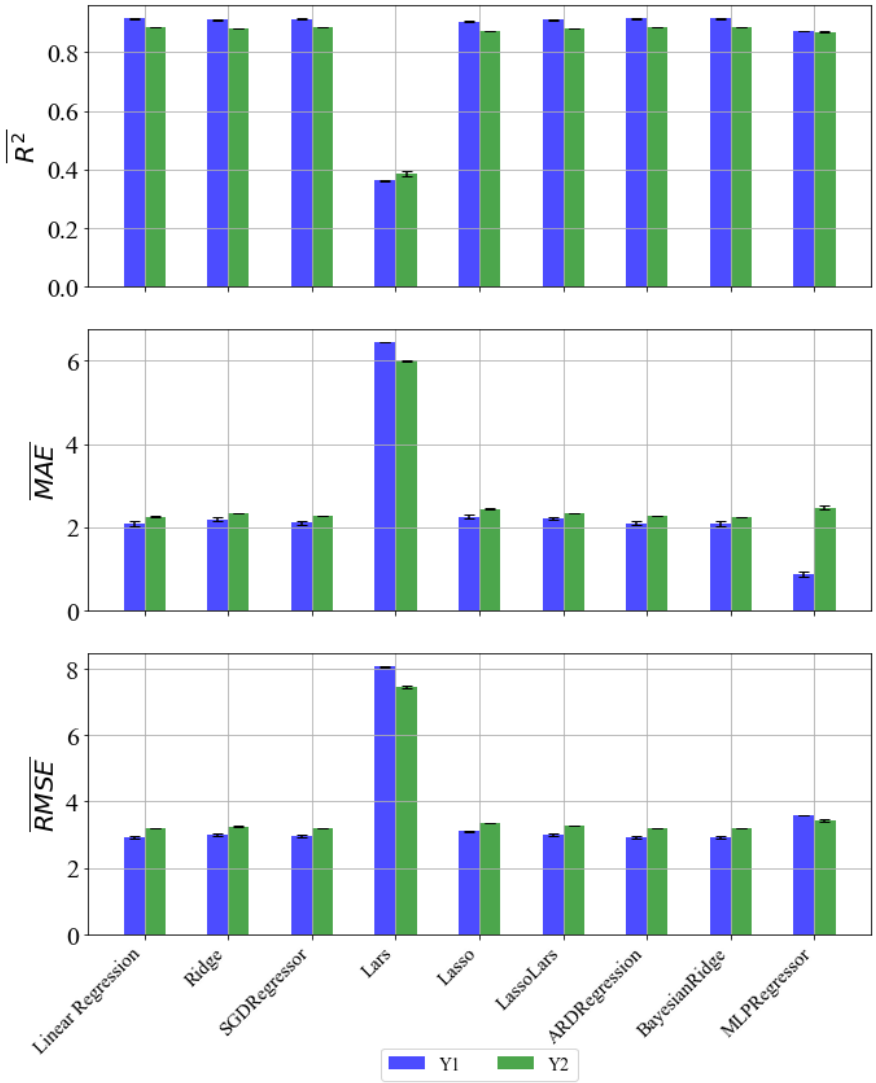
Normalizaciju definiramo kao transformaciju podataka na takav način da novi skup podataka ima srednju vrijednost 0 i vrijednost standardne devijacije 1, tj. tako da se novi podaci mogu raspodijeliti po normalnoj distribuciji (Gaussovoj raspodjeli) [29]. Metoda koja se koristi kako bi se dobila takva raspodjela podataka je „PowerTransformer“ metoda. U tu svrhu metoda primjenjuje dvije matematičke transformacije koje se razlikuju po tome mogu li raditi sa pozitivnim i negativnim vrijednostima (Yeo-Johnson) ili samo sa pozitivnim vrijednostima (Box-Cox) [30].

Zadnja metoda koja će se u radu koristiti je „Normalizer“. Kod ove metode se svaki uzorak (tj. svaki red u skupu podataka) koji ima barem jednu komponentu koja nije jednaka nuli skalira neovisno o drugim uzorcima tako da mu norma (L1, L2 ili max) poprimi vrijednost 1. Norma „L1“ postavlja sumu apsolutnih vrijednosti na 1, norma „L2“ postavlja duljinu vektora na 1, a norma „max“ postavlja maksimalnu apsolutnu vrijednost na 1 [31].

Naredbe skaliranja i normalizacije potrebno je primijeniti na ulaznim podacima (X1-X8) prije nego se skup podataka razdvoji na podatke za treniranje i podatke za testiranje.

5. INICIJALNA ANALIZA I REZULTATI

Kako bi se izgradili modeli za predikciju potrebno je prvo pripremiti skup podataka. Budući da postoje dvije ciljne varijable, toplinski dobitak i toplinsko opterećenje, potrebno je iz originalnog skupa podataka izvaditi jednu ciljnu varijabli kad gradimo model za predikciju druge. Kako bi se modeli istrenirali i testirali koristimo tehniku pod nazivom „train-test split“ s omjerom 70/30. Ovo znači da 70% preostalih podataka koristimo za treniranje modela, a 30% koristimo za testiranje. Zatim se primjenjuje 9 prethodno opisanih metoda za treniranje modela. Nakon što smo istrenirali modele provodimo evaluaciju njihovih performansi kako bi smo saznali koliko su precizni i pouzdani.



Slika 5.1. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – bez skaliranja i normalizacije

Na slici 5.1. prikazani su rezultati evaluacijskih metrika za modele dobivene korištenjem zadanih hiperparametara i bez skaliranja ili normalzacije skupa podataka.

Za ocjenu kvalitete našeg modela previđanja možemo koristiti više parametara. Uzmimo za primjer evaluacijsku metodu kvadratne greške (R^2). Prema definiciji rezultati dobiveni ovom metodom mogu postići vrijednosti od 0 do 1, pri čemu za veće vrijednosti smatramo da je ostvarena bolja regresijska analiza. Kod ove evaluacijske metode skoro svi algoritmi su dali zadovoljavajuće rezultate izuzev rezultata dobivenih korištenjem Lars metode, kod koje su dobivene preniske vrijednosti.

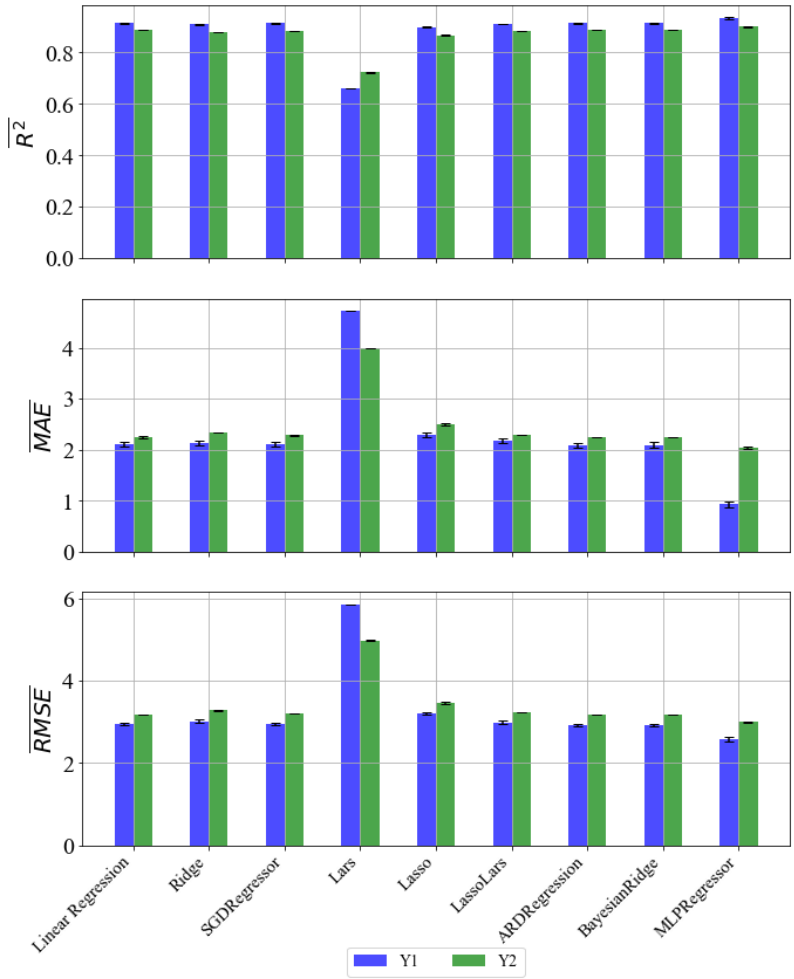
Za metodu srednje apsolutne greške algoritam smatramo kvalitetnijim što su dobiveni rezultati niži, a također je potrebno da su rezultati treniranja i testiranja međusobno što sličniji. Ove uvjete ispunjavaju svi algoritmi, a jedini koji se ističe je Lars koji daje nešto veće rezultate od ostalih.

Kod evaluacijske metode korijena srednje kvadratne greške rezultati dobiveni treniranim i testiranim skupom moraju imati što sličnije vrijednosti. Ovaj kriterij su zadovoljili svi algoritmi.

6. ANALIZA I REZULTATI DOBIVENI METODAMA SKALIRANJA I NORMALIZIRANJA

Na danom skupu podataka su pomoću prikladnih naredbi provedeni postupci skaliranja i normalizacije. Procedura kodiranja je započeta na način da se iz cjelokupnog skupa podataka (ulaznih i izlaznih varijabli) izdvoje samo ulazne varijable. Zatim su ulazne varijable skalirane i normalizirane nakon čega slijedi razdvajanje skupa podataka na dio za treniranje i dio za testiranje. Daljnji postupak je identičan kao i u poglavlju 5. Korištenjem 9 metodi trenirani su algoritmi za predviđanje izlaznih varijabli čija je kvaliteta analizirana korištenjem evaluacijskih metrika.

6.1. MaxAbsScaler



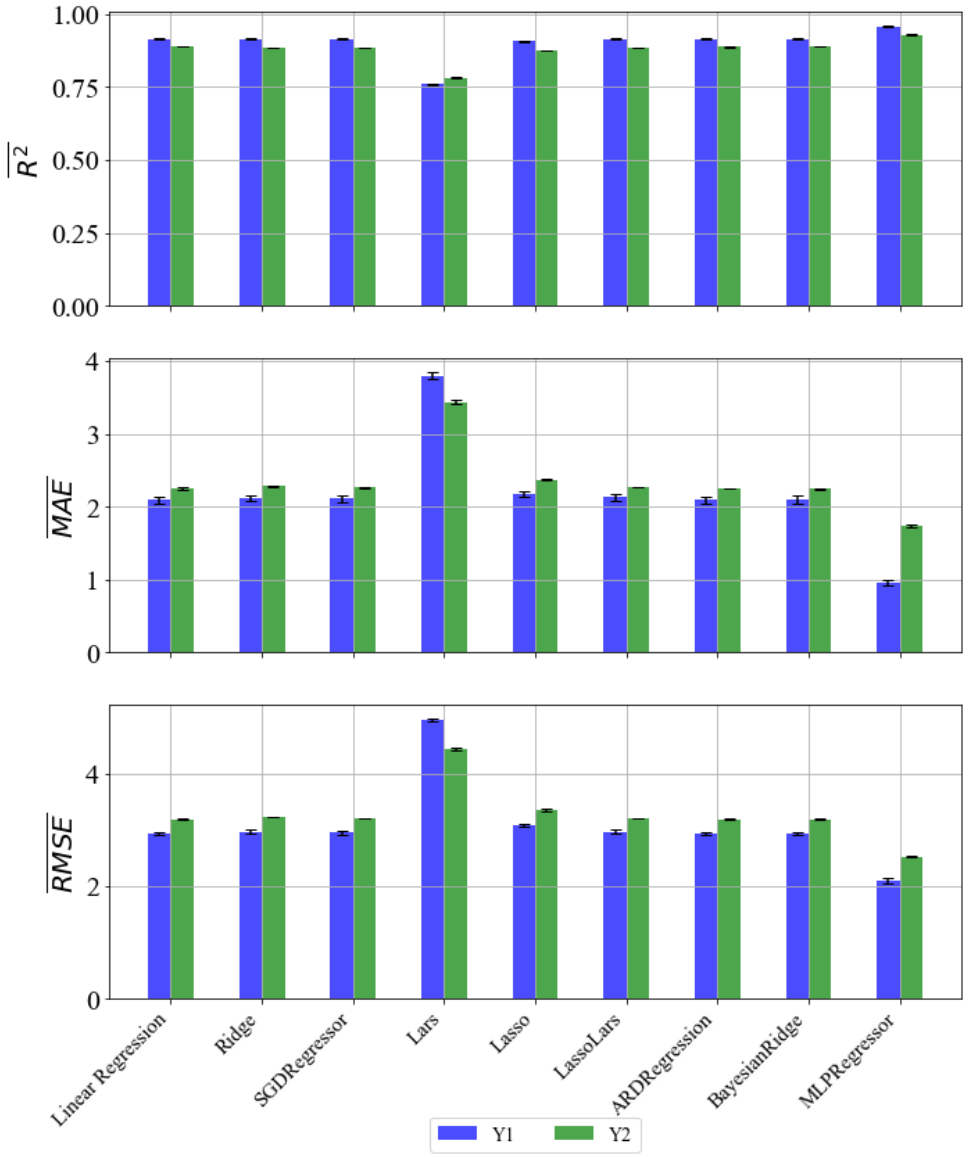
Slika 6.1. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – MaxAbsScaler

Evaluacijskom metrikom kvadrata greške (R^2) većina algoritama pokazuje kvalitetne rezultate. Kao i kod inicijalne analize najlošije rezultate daje Lars metoda, ali su oni u usporedbi sa rezultatima iz poglavlja 5 znatno bolji.

Gledajući rezultate dobivene metrikom srednje apsolutne greške Lars metoda ponovno daje nešto lošije rezultate, ali su također ponovno bolji nego kod inicijalne analize.

Kriterije evaluacijske metode korijena srednje kvadratne greške su zadovoljili svi algoritmi.

6.2. MinMaxScaler

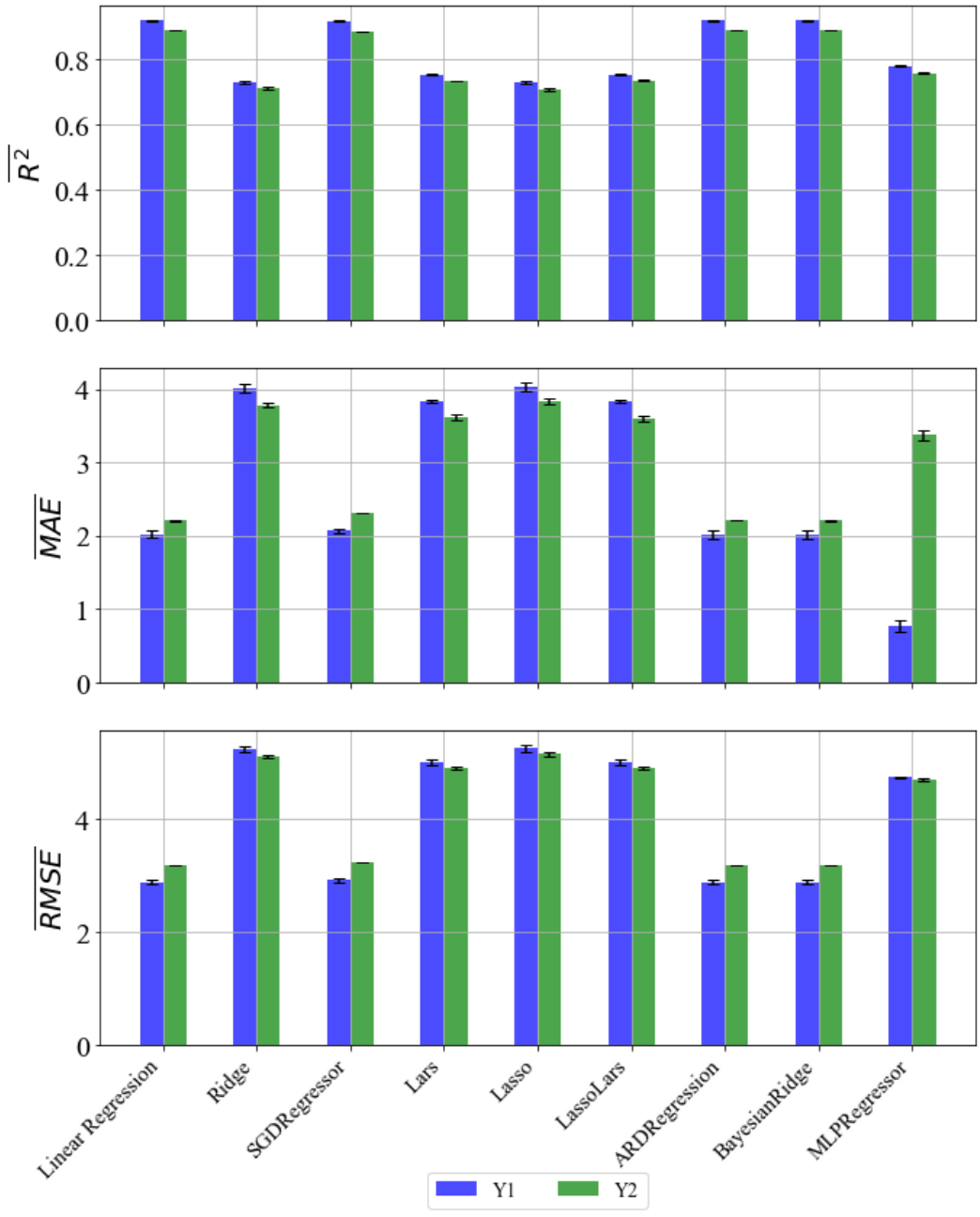


Slika 6.2. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – MinMaxScaler

Svim evaluacijskim metrikama dobiveni su slični ili bolji rezultati naspram rezultata dobivenih inicijalnom analizom.

R^2 evaluacija je pokazala bolje rezultate za Lars metodu nego što je dobiveno u slučaju gdje je korišten MaxAbsScaler.

6.3. Normalizer



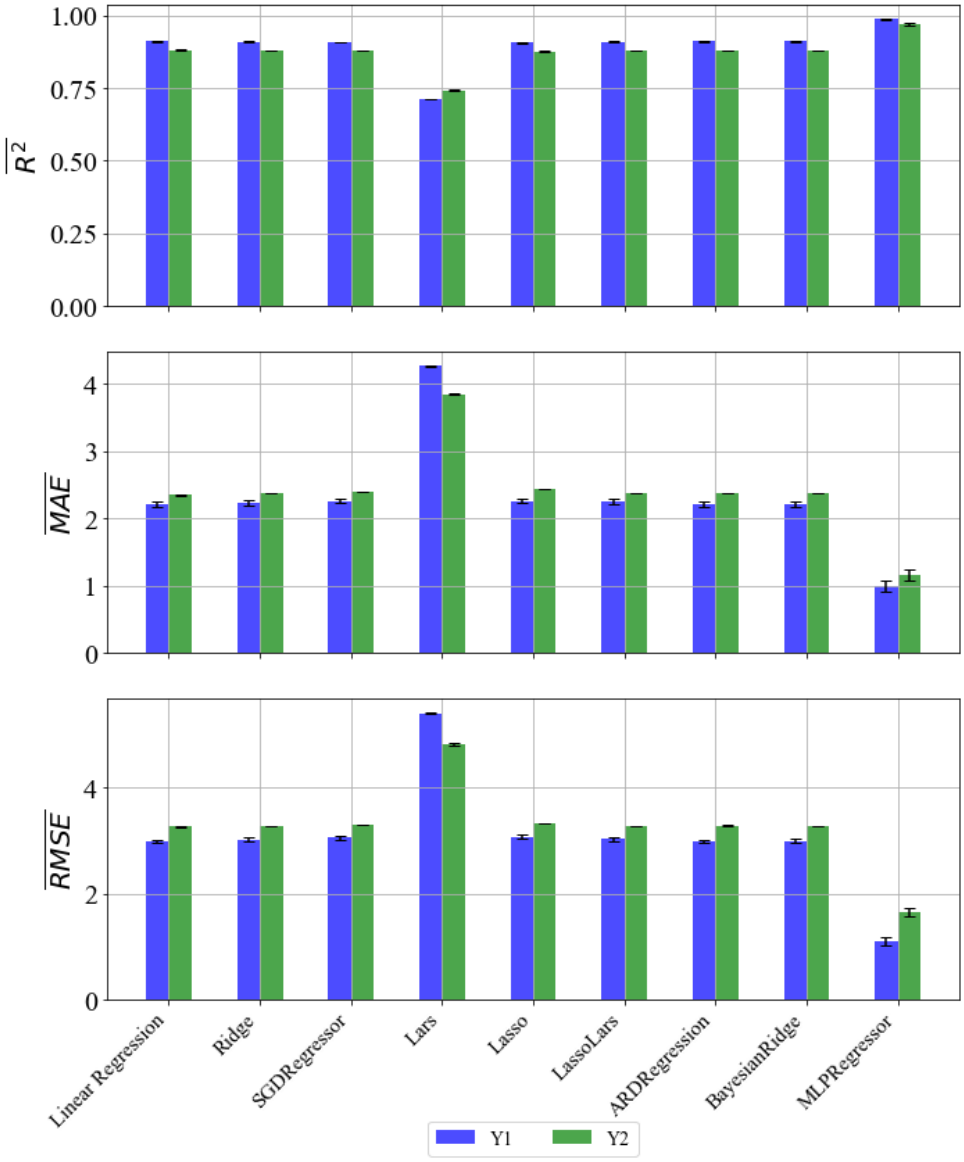
Slika 6.3. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – Normalizer

Evaluacijska metrika R^2 pokazala je lošije rezultate za metode Ridge, Lasso-Lars, Lasso i MLP Regressor nego što su inicijalno dobiveni. Lars metoda je ponovno dala nešto kvalitetnije rezultate nego što je dala u inicijalnoj analizi.

Evaluacijska metrika MAE pokazuje da su Ridge, Lasso, Lasso-Lars i MLP Regressor metode dale lošije rezultate nego inicijalno, a Lars bolje.

Sve metode zadovoljavaju kriterije RMSE evaluacijske metrike.

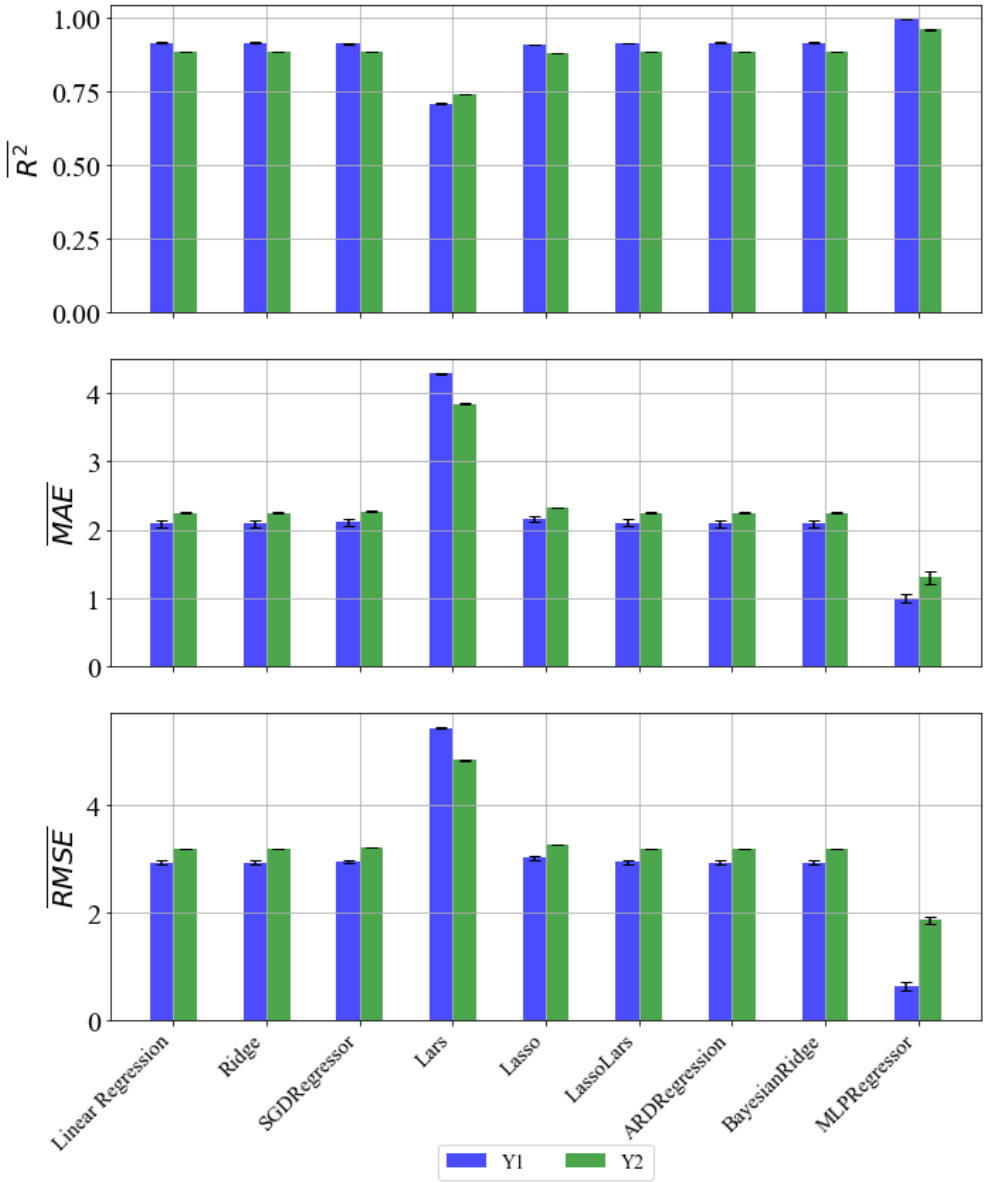
6.4. PowerTransformer



Slika 6.4. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – PowerTransformer

U usporedbi sa inicijalnom analizom bolje rezultate su dale Lars (za R^2 evaluacijsku metodu) i MLP Regressor (za R^2 i MAE evaluacijsku metodu). Ostale vrijednosti su slične ili marginalno različite od rezultata inicijalne analize.

6.5. StandardScaler



Slika 6.5. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – StandardScaler

U usporedbi sa inicijalnom analizom bolje rezultate su dale Lars i MLP Regressor (za R^2 i MAE evaluacijsku metodu). Ostale vrijednosti su slične ili marginalno različite od rezultata inicijalne analize.

7. NASUMIČNI ODABIR HIPERPARAMETARA

Algoritmi strojnog učenja uključuju određeni broj hiperparametara koji moraju biti postavljeni prije pokretanja. Hiperparametri su parametri koje u strojnom učenju možemo jednostavno definirati kao argumente funkcija ili kao parametre učenja. Uzmimo za primjer „LinearRegression“ metodu iz trećeg poglavlja ovoga rada. Kod te metode hiperparametri su redom: *fit_intercept, copy_X, n_jobs* i *positive*. Isti princip vrijedi i za sve ostale prethodno objašnjene metode. Ovi hiperparametri moraju biti oprezno optimizirani kako bi se postigle maksimalne performanse algoritma, odnosno kako bi se postizali dobri rezultati u čim kraćem vremenskom periodu. [24].

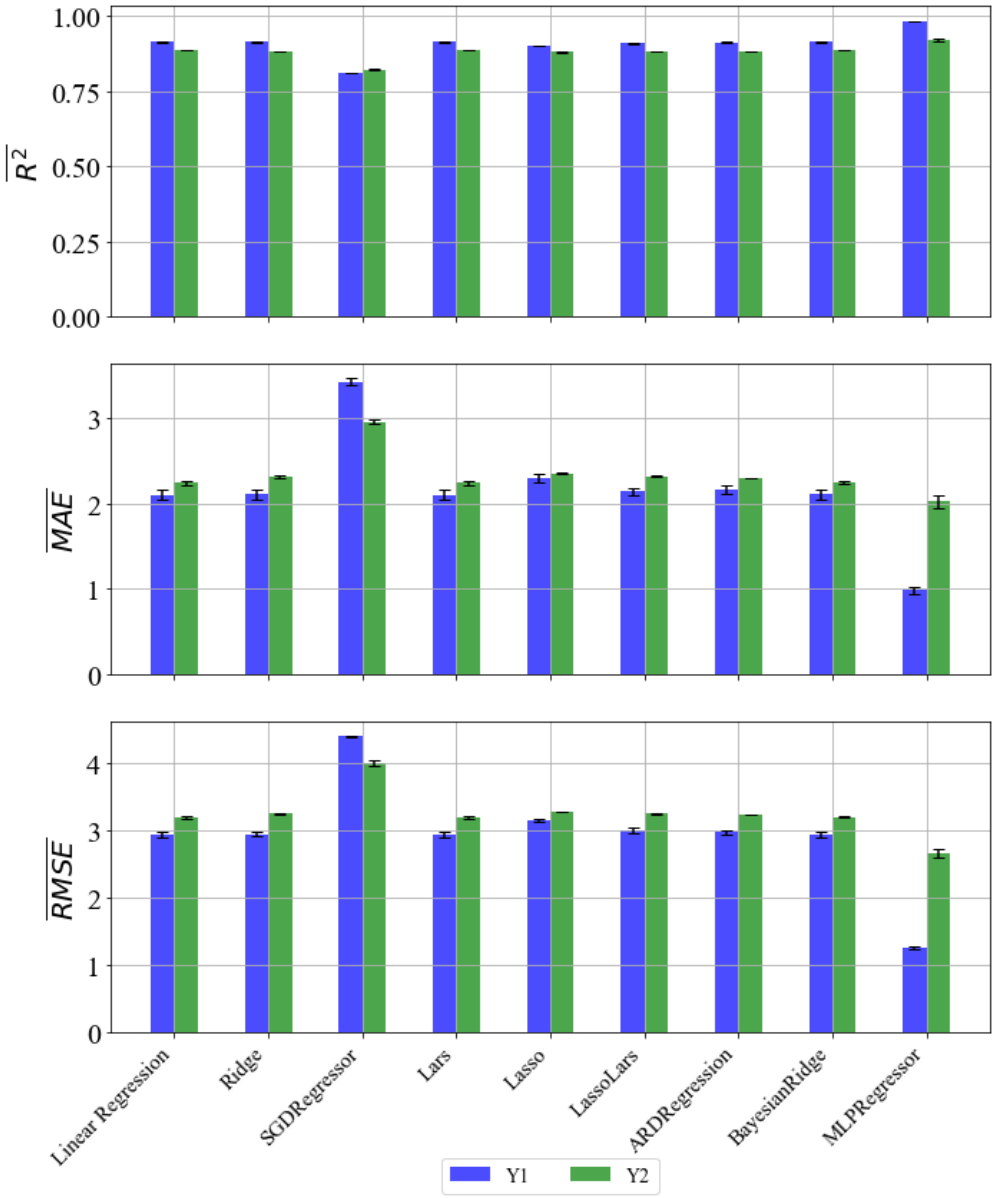
Kako bi se odabrala odgovarajuća konfiguracija hiperparametara za određeni skup podataka korisnici algoritama strojnog učenja mogu koristiti zadane vrijednosti hiperparametara ili ih ručno konfigurirati na temelju iskustva, preporuka iz literature, metodom pokušaja i pogreške i sl. Alternativni pristup dobivanja optimalne konfiguracije hiperparametara je korištenje strategije podešavanja hiperparametara (eng. hyperparameter tuning strategies). Kod takvih strategija algoritam samostalno odabire najbolju kombinaciju, odnosno onu kombinaciju koja će rezultirati najmanjom greškom. Ove strategije podešavanja variraju od jednostavnijih, kao što su *grid search* i *random search*, do kompleksnijih kao što je Bayesova optimizacija. Uz odabir efektivne strategije podešavanja korisnik mora odabrati odgovarajuće raspone i skale za skup hiperparametara. Loše odabrani rasponi mogu rezultirati smanjenjem kvalitete i učinkovitosti modela, te smanjenjem brzine konvergencije postupka ugađanja [24].

U ovom radu za nasumični odabir hiperparametara će se upotrijebiti *random search* metoda. Kod ove metode koriste se određene naredbe koje rade nasumičan odabir podataka ovisno o tipu tih podataka. Primjerice ako se radi cjelobrojnom podatku odabrati će se cijeli broj unutar zadanog raspona, ako se radi o *float* tipu podataka odabrati će se neki decimalni broj iz raspona, ukoliko se radi o *bool* tipu podataka odabrati će se vrijednost 1 ili 0, itd. Ukratko rečeno, *random search* metoda će isprobavati sve moguće kombinacije odabranih hiperparametara unutar raspona koji smo zadali [25]. Pretraživanje hiperparametara će biti zaustavljeno kada iznos srednje vrijednosti kvadratne greške poprimi zadovoljavajuću vrijednost.

U ovom poglavlju prikazati će se rezultati dobiveni podešavanjem hiperparametara korištenjem *random search* metode za svih 9 metoda koje su i prethodno korištene za treniranje algoritama

za predviđanje i to sa i bez primjene metoda skaliranja i normalizacije. U nastavku su prikazane tablice koje sadrže rezultate korištenih evaluacijskih metrika za nasumičan odabir hiperparametara i vrijednosti tih hiperparametara. Također je prikazan korak iteracije u kojemu su pronađena zadovoljavajuća rješenja. Rezultati će se uspoređivati sa rezultatima iz poglavlja 5. i poglavlja 6.

7.1. Rezultati dobiveni bez skaliranja i normalizacije



Slika 7.1. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – bez skaliranja i normalizacije

Usporedbom rezultata iz poglavlja 5. i poglavlja 7.1. vidimo da su rezultati za većinu metoda slični ili neznačajno bolji u slučaju nasumičnog odabira hiperparametara. Najviše se ističu metode Lars, Lasso i MLPRegressor. Za Lasso metodu dobiveni su primjetno bolji rezultati, ali najveća razlika se može primijetiti za metode Lars i MLPRegressor, jer su nasumičnim odabirom hiperparametara za obje metode dobiveni puno bolji rezultati za sve evaluacijske metrike.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 7.1.10. LinearRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	False	/	False
normalize	/	False	/	False
copy_X	/	True	/	False
n_jobs	10-85	72	1-100	67
positive	/	False	/	False

Tablica 7.1.11. Ridge – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-6	3.3092	0.1-10	2.0501
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
max_iter	100-1000	223	100-1000	288
tol	10^{-3}-10^{-2}	0.0018	10^{-4}-10^{-3}	0.0005
solver	/	sparse_cg	/	lsqr
random_state	20-100	47	40-100	68

Tablica 7.1.12. SGDRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.001-0.02	0.0038	0.004-0.01	0.0045
fit_intercept	/	True	/	True
max_iter	100-1000	257	100-1000	534
penalty	/	l1	/	l1
l1_ratio	0.2-0.5	0.3367	0.1-0.9	0.8625
shuffle	/	False	/	False

verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.2369	0.03-0.1	0.0484
learning_rate	/	adaptive	/	invscaling
eta0	0.01-0.5	0.2039	0.06-0.09	0.0868
power_t	0.2-1.0	0.7725	0.1-0.4	0.3011
early_stopping	/	True	/	False
tol	10^{-3}-10^{-2}	0.0084	10^{-5}-10^{-4}	9.53432e-05
random_state	1-10	4	0-60	34
validation_fraction	0.01-1.0	0.5052	0.01-1.0	0.3297
n_iter_no_change	1-10	9	1-10	9
warm_start	/	True	/	False
average	/	False	/	False

Tablica 7.1.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	False
verbose	/	True	/	True
normalize	/	True	/	False
precompute	/	True	/	auto
n_nonzero_coefs	50-90	52	5-9	7
eps	10^{-12}-10^{-10}	6.9098e-11	10^{-5}-10^{-4}	2.8249e-05
copy_X	/	True	/	True
fit_path	/	True	/	True
jitter	10^{-6}-10^{-4}	3.3955e-05	10^{-6}-$2*10^{-5}$	3.828e-06
random_state	30-80	70	1-50	22

Tablica 7.1.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.019	0.0121	0.0005-0.008	0.0011
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	False	/	False
copy_X	/	False	/	True
max_iter	100-1000	905	100-1000	787
tol	10^{-4}-10^{-3}	0.0009	10^{-5}-10^{-4}	7.7213e-05
warm_start	/	False	/	True
positive	/	False	/	False
random_state	30-80	66	40-80	74

Tablica 7.1.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.003-0.009	0.0059	0.01-0.1	0.0217
fit_intercept	/	True	/	False
verbose	/	True	/	True
normalize	/	False	/	True
precompute	/	True	/	auto
max_iter	100-1000	285	100-1000	183
eps	/	0.0007	/	0.0008
copy_X	/	True	/	True
fit_path	/	False	/	True
positive	/	False	/	False
jitter	0.04-0.1	0.0798	0.04-0.1	0.079
random_state	70-100	81	30-90	36

Tablica 7.1.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	485	200-350	214
tol	10^{-4}-10^{-3}	0.0002	10^{-6}-10^{-5}	5.0226e-06
alpha_1	10^{-4}-10^{-5}	9.6847e-05	10^{-6}-10^{-5}	8.6169e-06
alpha_2	10^{-6}-10^{-5}	5.7583e-06	10^{-7}-10^{-5}	3.8726e-07
lambda_1	10^{-7}-$4*10^{-7}$	1.3105e-07	10^{-8}-10^{-6}	9.2107e-07
lambda_2	10^{-6}-10^{-5}	6.9949e-06	10^{-6}-10^{-5}	9.8104e-06
compute_score	/	True	/	True
threshold_lambda	9800-10400	9884.7	9800-10500	10093.7
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
verbose	/	True	/	True

Tablica 7.1.17. BayesianRidge – Dobiveni hiperparametri

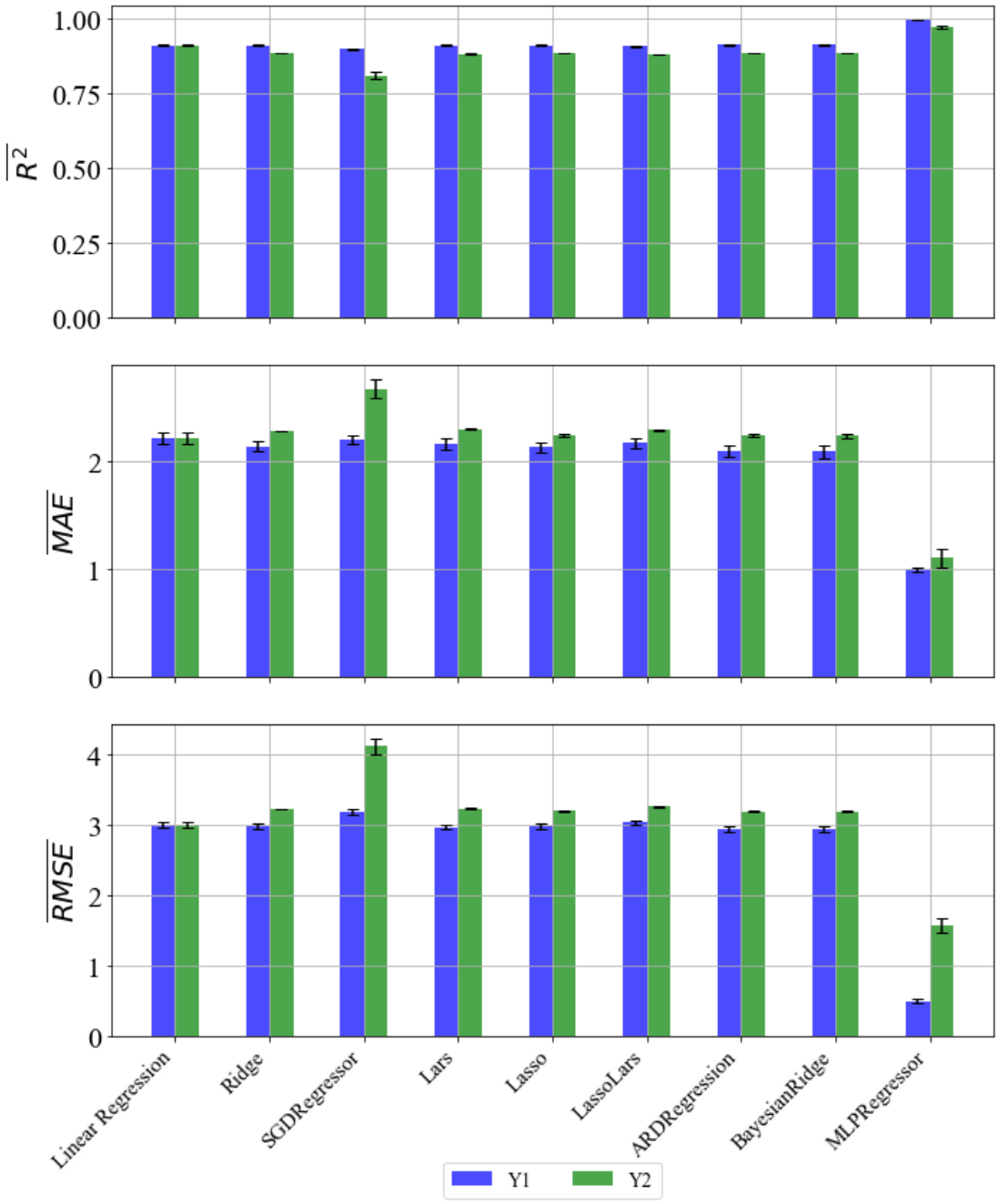
	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	372	300-500	395
tol	$2*10^{-3}$-10^{-3}	0.0019	$2*10^{-4}$-10^{-3}	0.0003
alpha_1	10^{-6}-10^{-5}	5.9657e-06	10^{-6}-$8*10^{-6}$	1.1309e-06
alpha_2	10^{-7}-10^{-5}	2.8309e-06	10^{-7}-10^{-5}	3.3191e-07
lambda_1	10^{-6}-10^{-5}	3.5059e-06	10^{-6}-$8*10^{-6}$	7.1184e-06
lambda_2	10^{-7}-10^{-5}	7.9309e-06	$5*10^{-7}$-10^{-5}	1.1324e-06
alpha_init	10^{-7}-10^{-4}	9.3658e-05	10^{-7}-$5*10^{-6}$	4.5871e-06

lambda_init	10^{-7}-10^{-5}	2.5911e-06	10^{-6}-10^{-5}	9.119e-06
compute_score	/	True	/	True
fit_intercept	/	True	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
verbose	/	False	/	True

Tablica 7.1.18. MLPRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(94,)	10-100	(68,)
activation	/	relu	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0007	10^{-5}-10^{-3}	7.9332e-05
batch_size	do 200	52	do 200	162
learning_rate	/	adaptive	/	constant
learning_rate_init	10^{-5}-10^{-4}	3.381e-05	10^{-5}-10^{-3}	0.0006
power_t	0.3-0.7	0.4365	0.1-0.9	0.4771
max_iter	300-500	400	200-500	461
shuffle	/	False	/	False
random_state	0-150	3	0-100	100
tol	10^{-5}-10^{-3}	0.0007	10^{-5}-10^{-3}	0.0004
verbose	/	True	/	True
warm_start	/	True	/	True
momentum	0.2-0.8	0.3777	0.1-0.9	0.7514
nesterovs_momentum	/	True	/	True
early_stopping	/	False	/	False
validation_fraction	0.1-0.3	0.2303	0.1-0.3	0.2898
beta_1	0.7-0.9	0.7414	0.7-0.9	0.773
beta_2	0.8-0.99	0.8328	0.8-0.99	0.9354
epsilon	10^{-9}-10^{-8}	7.4316e-09	10^{-9}-10^{-7}	2.0108e-08
n_iter_no_change	5-20	11	5-20	9
max_fun	10000-20000	17601	10000-20000	13219

7.2. Rezultati dobiveni korištenjem MaxAbsScaler-a



Slika 7.2. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – MaxAbsScaler

Usporedbom rezultata iz poglavlja 6.1. i poglavlja 7.2. vidimo da su rezultati za većinu metoda slični ili neznačajno bolji u slučaju nasumičnog odabira hiperparametara, kao i kod prethodnog slučaja. Najviše se ističu metode Lars, Lasso i MLPRegressor. Za Lasso metodu dobiveni su primjetno bolji rezultati, ali najveća razlika se može primijetiti za metode Lars i

MLPRegressor, jer su nasumičnim odabirom hiperparametara za obje metode dobiveni puno bolji rezultati za sve evaluacijske metrike.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 7.2.10. LinearRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	False	/	True
copy_X	/	True	/	True
n_jobs	0-10	2	1-100	90
positive	/	False	/	False

Tablica 7.2.11. Ridge – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-1.5	0.1002	0.08-0.1	0.0926
fit_intercept	/	False	/	True
normalize	/	True	/	False
copy_X	/	True	/	True
max_iter	10-100	31	100-1000	515
tol	10^{-6}-10^{-3}	5.6802e-05	10^{-7}-10^{-6}	9.7991e-07
solver	/	cholesky	/	sag
random_state	0-10	6	50-100	99

Tablica 7.2.12. SGDRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.001-0.001	0.0002	0.001-0.005	0.0022
fit_intercept	/	True	/	True
max_iter	100-1000	607	100-1000	887
penalty	/	l1	/	l1
l1_ratio	0.0-0.3	0.2519	0.2-0.9	0.2261
shuffle	/	False	/	False
verbose	0-0	0	0-0	0
epsilon	0.2-1.0	0.7281	0.03-0.08	0.0375
learning_rate	/	adaptive	/	adaptive
eta0	0.15-0.35	0.2908	0.05-0.1	0.0876
power_t	0.1-0.7	0.4296	0.1-0.5	0.2361

early_stopping	/	False	/	False
tol	10^{-3}-10^{-2}	0.0082	10^{-4}-10^{-3}	0.0005
random_state	3-7	3	10-60	55
validation_fraction	0.01-1.0	0.0385	0.01-1.0	0.8103
n_iter_no_change	1-10	7	1-10	8
warm_start	/	True	/	True
average	/	False	/	False

Tablica 7.2.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
verbose	/	False	/	True
normalize	/	False	/	False
precompute	/	True	/	auto
n_nonzero_coefs	13-85	57	5-15	10
eps	10^{-12}-10^{-10}	6.6372e-11	10^{-6}-10^{-4}	6.945e-05
copy_X	/	True	/	True
fit_path	/	False	/	True
jitter	10^{-7}-10^{-4}	7.4177e-05	10^{-5}-10^{-4}	5.1382e-05
random_state	10-100	65	3-35	27

Tablica 7.2.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.008-0.02	0.0086	0.0001-0.001	0.0001
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	True	/	False
copy_X	/	True	/	True
max_iter	100-1000	254	100-1000	339
tol	10^{-4}-10^{-3}	0.0008	10^{-4}-10^{-3}	0.0004
warm_start	/	False	/	False
positive	/	False	/	False
random_state	20-82	70	10-80	66

Tablica 7.2.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.09	0.0177	0.01-0.2	0.0101
fit_intercept	/	True	/	False
verbose	/	False	/	False
normalize	/	False	/	False
precompute	/	True	/	False
max_iter	100-1000	134	100-1000	716
eps	/	0.0003	/	0.0009
copy_X	/	True	/	True
fit_path	/	True	/	True
positive	/	False	/	False
jitter	0.02-0.1	0.0539	0.05-0.1	0.0983
random_state	40-100	54	30-70	48

Tablica 7.2.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	260-350	272	150-300	158
tol	10^{-5}-10^{-3}	0.0009	10^{-6}-10^{-5}	9.9705e-06
alpha_1	10^{-2}-10^{-4}	0.0048	10^{-6}-10^{-5}	2.6908e-06
alpha_2	10^{-7}-10^{-5}	4.7752e-06	10^{-6}-10^{-5}	9.5248e-06
lambda_1	10^{-7}-10^{-6}	5.0997e-07	10^{-7}-10^{-6}	7.4595e-07
lambda_2	10^{-6}-10^{-5}	8.7896e-06	10^{-6}-10^{-5}	8.4708e-06
compute_score	/	True	/	True
threshold_lambda	9300-10500	10212.6	9000-11000	10525.1
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
verbose	/	True	/	True

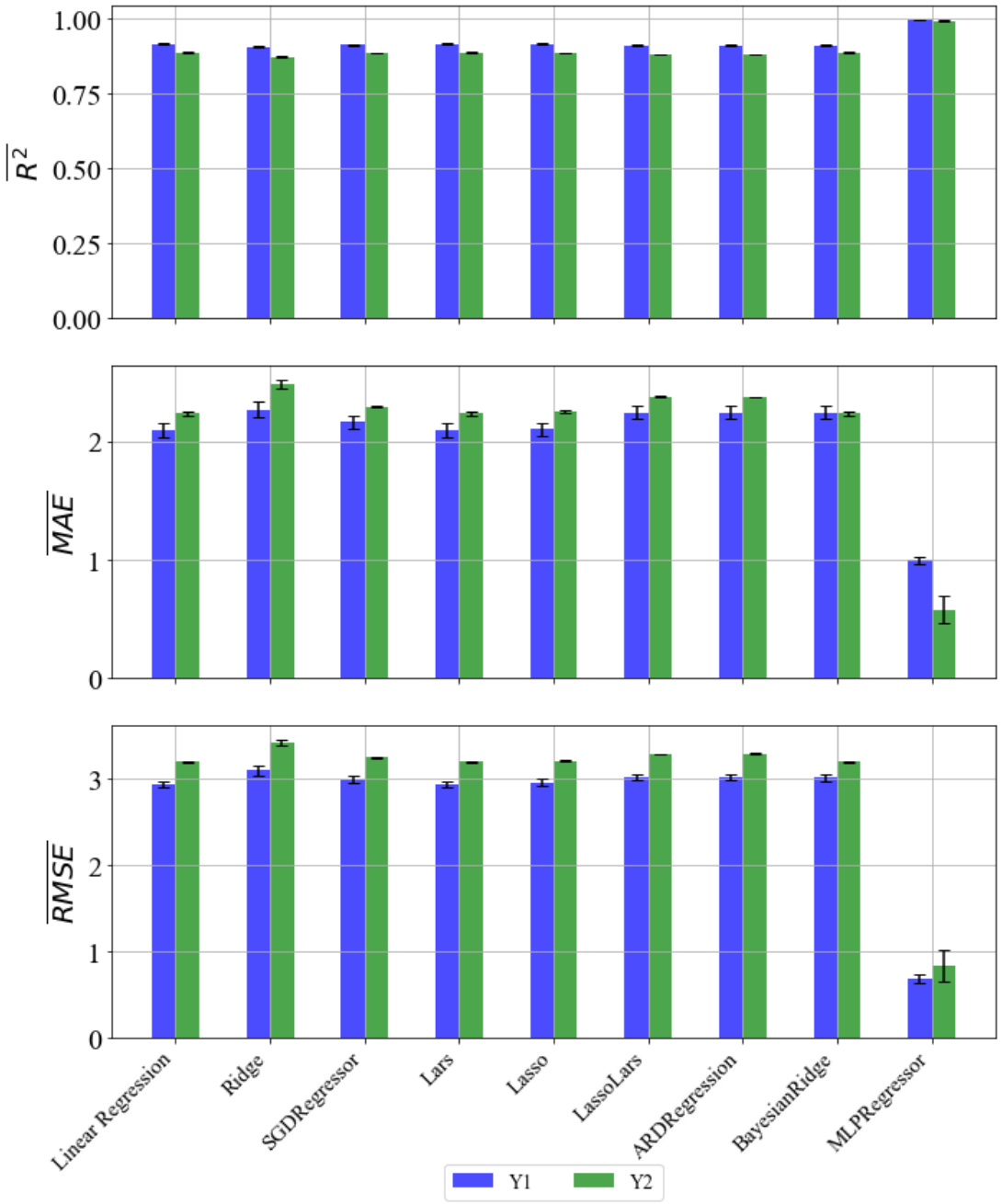
Tablica 7.2.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	391	300-500	307
tol	10^{-}-10^{-2}	0.0016	10^{-4}-10^{-3}	0.0007
alpha_1	10^{-6}-$7*10^{-6}$	2.2264e-06	10^{-7}-10^{-5}	9.0205e-06
alpha_2	$2*10^{-6}$-$7*10^{-6}$	4.2899e-06	10^{-7}-10^{-5}	9.6441e-07
lambda_1	$2*10^{-6}$-$8*10^{-6}$	4.6369e-06	10^{-7}-10^{-5}	1.6741e-06
lambda_2	$4*10^{-7}$-10^{-5}	3.8483e-07	10^{-7}-10^{-5}	3.4325e-06
alpha_init	$2*10^{-7}$-10^{-6}	4.2037e-06	10^{-7}-10^{-5}	2.4995e-06
lambda_init	10^{-7}-$2*10^{-6}$	6.0951e-07	10^{-7}-10^{-5}	7.7334e-06
compute_score	/	True	/	True
fit_intercept	/	True	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
verbose	/	False	/	False

Tablica 7.2.18. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(16, 28, 52)	10-100	(97, 72)
activation	/	tanh	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	9.0572e-05	10^{-5}-10^{-3}	0.0005
batch_size	do 200	25	do 200	125
learning_rate	/	adaptive	/	invscaling
learning_rate_init	10^{-5}-10^{-4}	1.7134e-05	10^{-5}-10^{-3}	0.0002
power_t	0.3-0.7	0.6101	0.1-0.9	0.4488
max_iter	300-500	401	200-500	290
shuffle	/	False	/	True
random_state	0-150	42	0-100	100
tol	10^{-5}-10^{-3}	0.0002	10^{-5}-10^{-3}	0.0008
verbose	/	True	/	True
warm_start	/	True	/	False
momentum	0.2-0.8	0.6839	0.1-0.9	0.1775
nesterovs_momentum	/	False	/	False
early_stopping	/	True	/	False
validation_fraction	0.1-0.3	0.2489	0.1-0.3	0.2324
beta_1	0.7-0.9	0.8001	0.7-0.9	0.7116
beta_2	0.8-0.99	0.9253	0.8-0.99	0.8488
epsilon	10^{-9}-10^{-8}	5.6573e-09	10^{-9}-10^{-7}	3.7957e-08
n_iter_no_change	5-20	18	5-20	14
max_fun	10000-20000	15931	10000-20000	14406

7.3. Rezultati dobiveni korištenjem MinMaxScaler-a



Slika 7.3. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – MinMaxScaler

Usporedbom rezultata iz poglavlja 6.2. i poglavlja 7.3. vidimo da su rezultati za većinu metoda slični ili neznačajno bolji u slučaju nasumičnog odabira hiperparametara. Najviše se ističu metode Lars, Lasso, LassoLars i MLPRegressor. Za Lasso i LassoLars metode dobiveni su primjetno bolji rezultati, ali najveća razlika se može primijetiti za metode Lars i

MLPRegressor, jer su nasumičnim odabirom hiperparametara za obje metode dobiveni puno bolji rezultati za sve evaluacijske metrike.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 7.3.10. LinearRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	False	/	False
normalize	/	False	/	False
copy_X	/	True	/	False
n_jobs	10-85	72	1-100	67
positive	/	False	/	False

Tablica 7.3.11. Ridge – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-6	3.3092	0.1-10	2.0501
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
max_iter	100-1000	223	100-1000	288
tol	10^{-3}-10^{-2}	0.0018	10^{-4}-10^{-3}	0.0005
solver	/	sparse_cg	/	lsqr
random_state	20-100	47	40-100	68

Tablica 7.3.12. SGDRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.001-0.02	0.0038	0.004-0.01	0.0045
fit_intercept	/	True	/	True
max_iter	100-1000	257	100-1000	534
penalty	/	l1	/	l1
l1_ratio	0.2-0.5	0.3367	0.1-0.9	0.8625
shuffle	/	False	/	False
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.2369	0.03-0.1	0.0484
learning_rate	/	adaptive	/	invscaling
eta0	0.01-0.5	0.2039	0.06-0.09	0.0868
power_t	0.2-1.0	0.7725	0.1-0.4	0.3011

early_stopping	/	True	/	False
tol	10^{-3}-10^{-2}	0.0084	10^{-5}-10^{-4}	9.5343e-05
random_state	1-10	4	0-60	34
validation_fraction	0.01-1.0	0.5052	0.01-1.0	0.3297
n_iter_no_change	1-10	9	1-10	9
warm_start	/	True	/	False
average	/	False	/	False

Tablica 7.3.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	False
verbose	/	True	/	True
normalize	/	True	/	False
precompute	/	True	/	auto
n_nonzero_coefs	50-90	52	5-9	7
eps	10^{-12}-10^{-10}	6.9098e-11	10^{-5}-10^{-4}	2.8249e-05
copy_X	/	True	/	True
fit_path	/	True	/	True
jitter	10^{-6}-10^{-4}	3.3955e-05	10^{-6}-$2*10^{-5}$	3.828e-06
random_state	30-80	70	1-50	22

Tablica 7.3.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.019	0.0121	0.0005-0.008	0.0011
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	False	/	False
copy_X	/	False	/	True
max_iter	100-1000	905	100-1000	787
tol	10^{-4}-10^{-3}	0.0009	10^{-5}-10^{-4}	7.7212e-05
warm_start	/	False	/	True
positive	/	False	/	False
random_state	30-80	66	40-80	74

Tablica 7.3.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.003-0.009	0.0059	0.01-0.1	0.0217
fit_intercept	/	True	/	False
verbose	/	True	/	True
normalize	/	False	/	True
precompute	/	True	/	auto
max_iter	100-1000	285	100-1000	183
eps	/	0.0006	/	0.0007
copy_X	/	True	/	True
fit_path	/	False	/	True
positive	/	False	/	False
jitter	0.04-0.1	0.0798	0.04-0.1	0.079
random_state	70-100	81	30-90	36

Tablica 7.3.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	485	200-350	214
tol	10^{-4}-10^{-3}	0.0001	10^{-6}-10^{-5}	5.0225e-06
alpha_1	10^{-4}-10^{-5}	9.6846e-05	10^{-6}-10^{-5}	8.6169e-06
alpha_2	10^{-6}-10^{-5}	5.7582e-06	10^{-7}-10^{-5}	3.8726e-07
lambda_1	10^{-7}-$4*10^{-7}$	1.3105e-07	10^{-8}-10^{-6}	9.2107e-07
lambda_2	10^{-6}-10^{-5}	6.9948e-06	10^{-6}-10^{-5}	9.8103e-06
compute_score	/	True	/	True
threshold_lambda	9800-10400	9884.6	9800-10500	10093.7
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
verbose	/	True	/	True

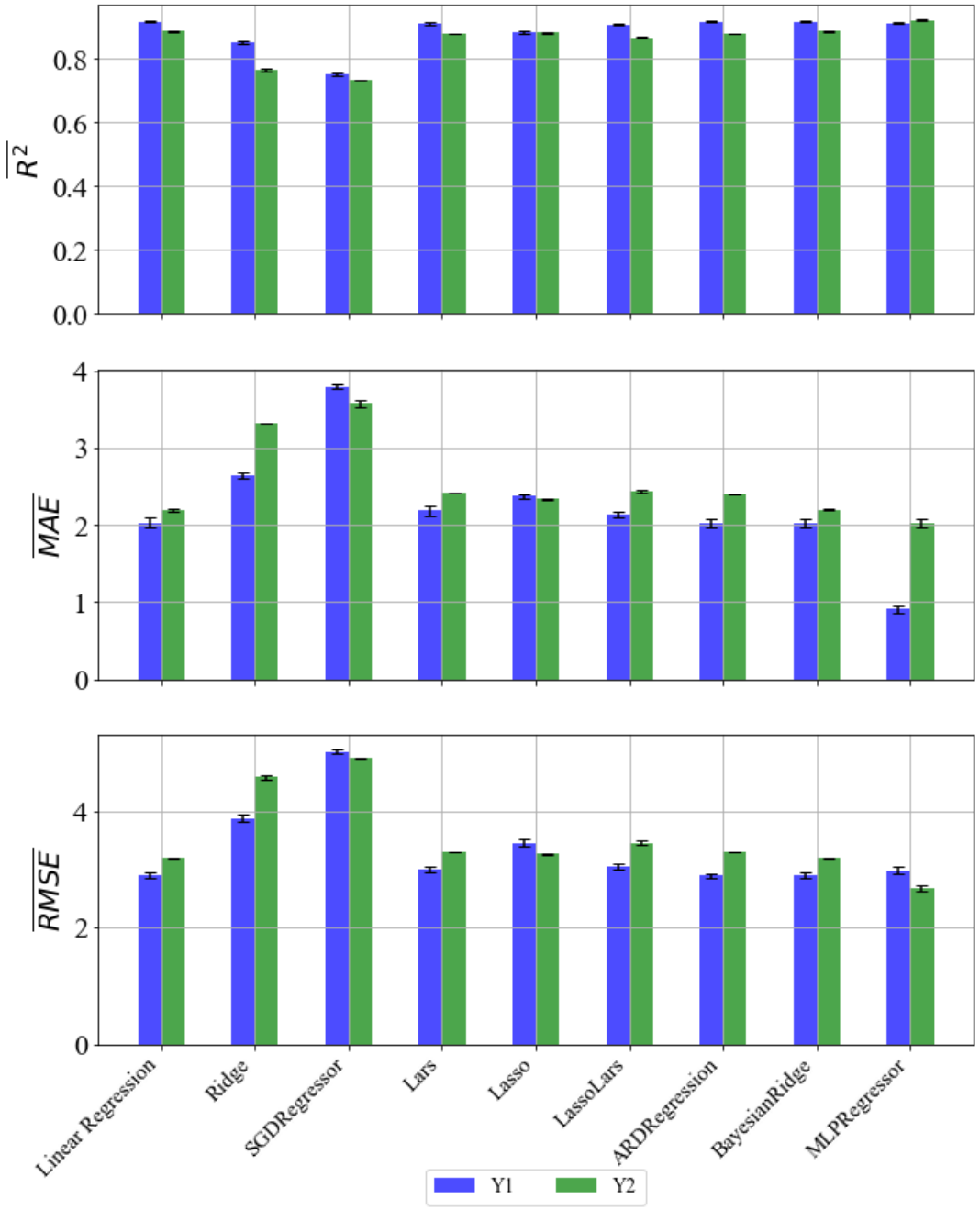
Tablica 7.3.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	372	300-500	395
tol	$2*10^{-3}$-10^{-3}	0.0019	$2*10^{-4}$-10^{-3}	0.0002
alpha_1	10^{-6}-10^{-5}	5.9656e-06	10^{-6}-$8*10^{-6}$	1.1308e-06
alpha_2	10^{-7}-10^{-5}	2.8308e-06	10^{-7}-10^{-5}	3.3190e-07
lambda_1	10^{-6}-10^{-5}	3.5058e-06	10^{-6}-$8*10^{-6}$	7.1183e-06
lambda_2	10^{-7}-10^{-5}	7.9309e-06	$5*10^{-7}$-10^{-5}	1.1324e-06
alpha_init	10^{-7}-10^{-4}	9.3658e-05	10^{-7}-$5*10^{-6}$	4.587e-06
lambda_init	10^{-7}-10^{-5}	2.5910e-06	10^{-6}-10^{-5}	9.119e-06
compute_score	/	True	/	True
fit_intercept	/	True	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
verbose	/	False	/	True

Tablica 7.3.18. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(94,)	10-100	(68,)
activation	/	relu	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0007	10^{-5}-10^{-3}	7.9331e-05
batch_size	do 200	52	do 200	162
learning_rate	/	adaptive	/	constant
learning_rate_init	10^{-5}-10^{-4}	3.381e-05	10^{-5}-10^{-3}	0.0005
power_t	0.3-0.7	0.4364	0.1-0.9	0.4771
max_iter	300-500	400	200-500	461
shuffle	/	False	/	False
random_state	0-150	3	0-100	100
tol	10^{-5}-10^{-3}	0.0007	10^{-5}-10^{-3}	0.0004
verbose	/	True	/	True
warm_start	/	True	/	True
momentum	0.2-0.8	0.3776	0.1-0.9	0.7513
nesterovs_momentum	/	True	/	True
early_stopping	/	False	/	False
validation_fraction	0.1-0.3	0.2303	0.1-0.3	0.2897
beta_1	0.7-0.9	0.7414	0.7-0.9	0.773
beta_2	0.8-0.99	0.8327	0.8-0.99	0.9353
epsilon	10^{-9}-10^{-8}	7.4316e-09	10^{-9}-10^{-7}	2.0107e-08
n_iter_no_change	5-20	11	5-20	9
max_fun	10000-20000	17601	10000-20000	13219

7.4. Rezultati dobiveni korištenjem Normalizer-a



Slika 7.4. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – Normalizer

Usporedbom rezultata iz poglavlja 6.3. i poglavlja 7.4. vidimo da su rezultati za većinu metoda slični ili neznačajno bolji u slučaju nasumičnog odabira hiperparametara. Najviše se ističu metode Ridge, Lars, Lasso, LassoLars i MLPRegressor. Za Ridge, Lasso i LassoLars metode dobiveni su dosta bolji rezultati, ali najveća razlika se može primijetiti za metode Lars

i MLPRegressor, jer su nasumičnim odabirom hiperparametara za obje metode dobiveni puno bolji rezultati za sve evaluacijske metrike.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 7.4.10. LinearRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
n_jobs	10-50	24	1-100	65
positive	/	False	/	False

Tablica 7.4.11. Ridge – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-0.7	0.2171	1.0-1.04	1.0338
fit_intercept	/	True	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
max_iter	100-1000	226	100-1000	514
tol	10^{-3}-10^{-2}	0.0014	10^{-4}-$5*10^{-4}$	0.0001
solver	/	auto	/	sag
random_state	5-30	16	80-100	89

Tablica 7.4.12. SGDRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.00001-0.1	0.0138	0.00001-0.1	0.0007
fit_intercept	/	True	/	True
max_iter	900-3000	2897	900-3000	774
penalty	/	l1	/	l1
l1_ratio	0.01-0.05	0.0233	0.01-0.05	0.3414
shuffle	/	False	/	False
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.9123	0.1-0.9	0.1
learning_rate	/	adaptive	/	invscaling
eta0	0.001-0.1	0.0855	0.001-0.1	0.0871
power_t	0.1-0.5	0.3415	0.1-0.5	0.2766

early_stopping	/	False	/	False
tol	10^{-3}-10^{-2}	0.0004	10^{-3}-10^{-2}	0.0004
random_state	0-10	3	0-10	67
Validation_fraction	0.01-1.0	0.1322	0.01-1.0	0.6195
warm_start	/	True	/	True
average	/	False	/	False
n_iter_no_change	1-10	6	1-10	6

Tablica 7.4.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	False
verbose	/	True	/	True
normalize	/	False	/	True
precompute	/	False	/	True
n_nonzero_coefs	100-800	508	80-120	94
eps	10^{-5}-10^{-4}	4.3111e-05	10^{-10}-10^{-3}	0.0005
copy_X	/	True	/	True
fit_path	/	False	/	False
jitter	10^{-5}-10^{-4}	2.2016e-05	10^{-5}-10^{-3}	0.0007
random_state	10-90	36	70-90	84

Tablica 7.4.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.03	0.0128	0.0004-0.003	0.0029
fit_intercept	/	True	/	True
normalize	/	True	/	True
precompute	/	True	/	True
copy_X	/	False	/	True
max_iter	100-1000	876	400-1000	662
tol	10^{-6}-10^{-3}	0.0004	10^{-5}-10^{-3}	0.0003
warm_start	/	False	/	True
positive	/	False	/	False
random_state	20-90	32	10-100	50

Tablica 7.4.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.00-0.01	0.0045	0.005-0.01	0.005
fit_intercept	/	True	/	True
verbose	/	False	/	False
normalize	/	True	/	True
precompute	/	auto	/	auto
max_iter	100-1000	201	100-1000	822
eps	/	0.0401	/	0.0002
copy_X	/	True	/	True
fit_path	/	True	/	False
positive	/	False	/	False
jitter	0.01-0.1	0.0979	0.01-0.1	0.0133
random_state	1-100	56	50-100	74

Tablica 7.4.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	297	150-350	314
tol	10^{-4}-10^{-3}	0.0005	10^{-6}-10^{-5}	2.0878e-06
alpha_1	10^{-4}-10^{-3}	0.0004	10^{-6}-10^{-5}	8.1278e-06
alpha_2	10^{-6}-10^{-5}	1.2863e-06	10^{-7}-10^{-6}	4.1525e-07
lambda_1	10^{-7}-10^{-6}	3.7281e-07	10^{-7}-10^{-6}	1.9953e-07
lambda_2	10^{-6}-10^{-5}	5.7832e-06	10^{-6}-10^{-5}	2.1341e-06
compute_score	/	True	/	True
threshold_lambda	9000-11000	10013.6	9000-10000	9195.6
fit_intercept	/	True	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
verbose	/	False	/	True

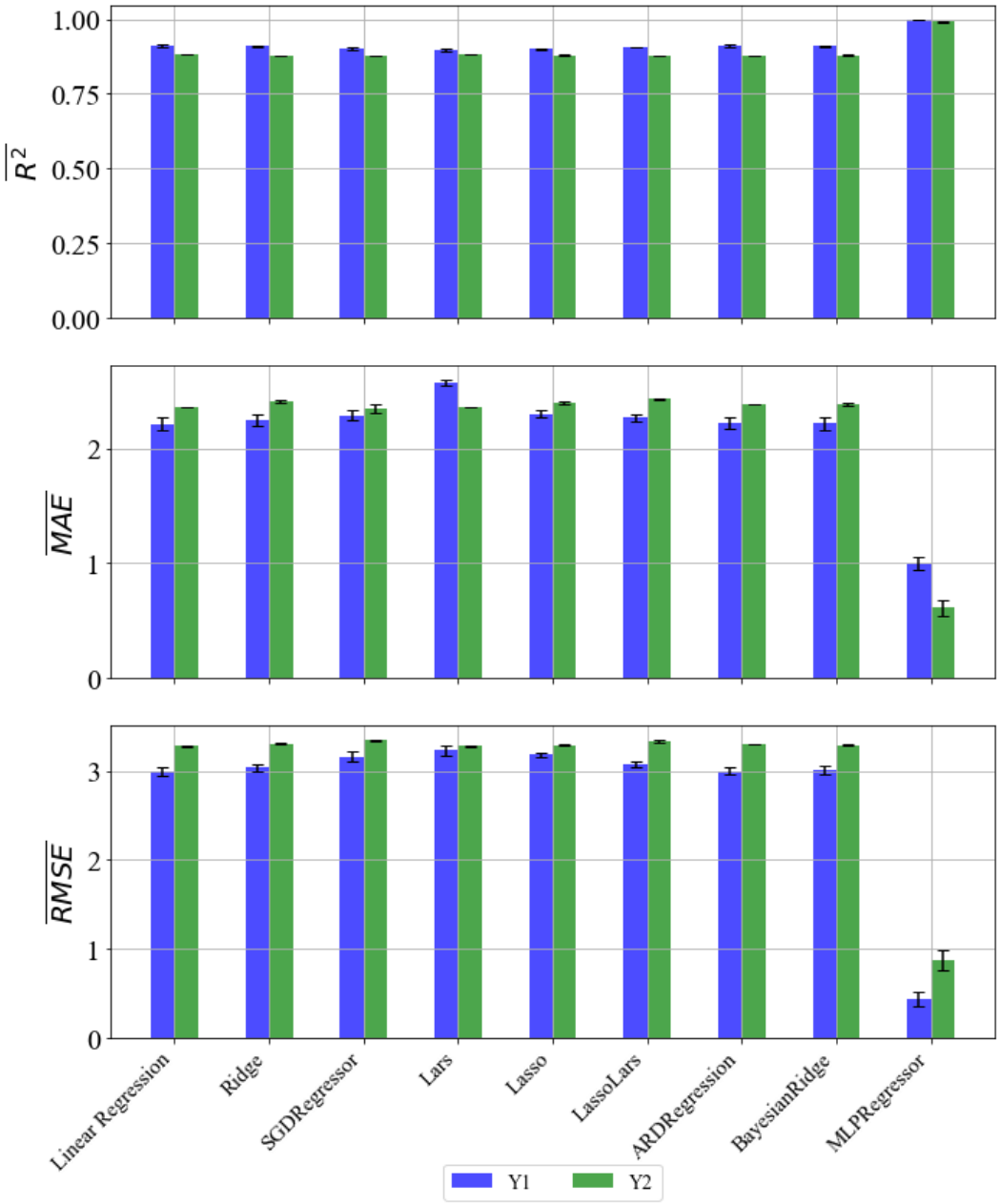
Tablica 7.4.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	340	300-500	436
tol	10^{-4}-10^{-2}	0.0064	10^{-4}-10^{-3}	0.0007
alpha_1	10^{-7}-10^{-5}	1.8765e-06	10^{-6}-10^{-5}	1.0895e-06
alpha_2	10^{-7}-10^{-5}	5.6694e-06	$2*10^{-6}$-10^{-5}	5.4544e-06
lambda_1	10^{-7}-10^{-5}	9.0157e-06	$5*10^{-6}$-10^{-5}	5.3682e-06
lambda_2	10^{-7}-10^{-5}	2.525e-06	$8*10^{-6}$-10^{-5}	9.8837e-06
alpha_init	10^{-7}-10^{-5}	7.5896e-06	10^{-7}-10^{-5}	7.2195e-06
lambda_init	10^{-7}-10^{-5}	7.5761e-06	$2*10^{-6}$-10^{-5}	5.0027e-06
compute_score	/	False	/	False
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
verbose	/	False	/	True

Tablica 7.4.18. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(48, 91, 48, 40)	10-100	(54, 69, 53, 96)
activation	/	identity	/	logistic
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0001	10^{-5}-10^{-3}	0.0004
batch_size	do 200	145	do 200	56
learning_rate	/	constant	/	invscaling
learning_rate_init	10^{-5}-10^{-4}	7.6222e-05	10^{-5}-10^{-3}	0.0002
power_t	0.3-0.7	0.5641	0.1-0.9	0.8876
max_iter	300-500	370	200-500	497
shuffle	/	True	/	False
random_state	0-150	129	0-100	27
tol	10^{-5}-10^{-3}	0.0003	10^{-5}-10^{-3}	0.0002
verbose	/	False	/	True
warm_start	/	False	/	False
momentum	0.2-0.8	0.2754	0.1-0.9	0.812
nesterovs_momentum	/	False	/	True
early_stopping	/	False	/	True
validation_fraction	0.1-0.3	0.1666	0.1-0.3	0.1761
beta_1	0.7-0.9	0.8578	0.7-0.9	0.8121
beta_2	0.8-0.99	0.8122	0.8-0.99	0.9614
epsilon	10^{-9}-10^{-8}	5.4267e-09	10^{-9}-10^{-7}	6.8162e-08
n_iter_no_change	5-20	15	5-20	9
max_fun	10000-20000	13320	10000-20000	16446

7.5. Rezultati dobiveni korištenjem PowerTransformer-a



Slika 7.5. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – PowerTransformer

Usporedbom rezultata iz poglavlja 6.4. i poglavlja 7.5. vidimo da su rezultati za većinu metoda slični ili neznačajno bolji u slučaju nasumičnog odabira hiperparametara. Najviše se ističu metode Lars, Lasso i MLPRegressor. Za Lasso i MLPRegressor metode dobiveni su nešto bolji rezultati, ali najveća razlika se može primijetiti kod metode Lars, jer su nasumičnim odabirom hiperparametara metode dobiveni puno bolji rezultati za sve evaluacijske metrike.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 7.5.10. LinearRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	False	/	True
copy_X	/	False	/	True
n_jobs	1-100	10	1-100	11
positive	/	False	/	False

Tablica 7.5.11. Ridge – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-0.7	0.3447	2.0-5.0	2.4267
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	False	/	True
max_iter	100-1000	751	100-1000	141
tol	10^{-3}-10^{-2}	0.0075	10^{-4}-10^{-3}	0.0003
solver	/	sag	/	svd
random_state	5-30	46	30-80	55

Tablica 7.5.12. SGDRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.00001-0.1	0.0638	0.00001-0.1	0.0257
fit_intercept	/	True	/	True
max_iter	900-3000	357	900-3000	401
penalty	/	l1	/	l2
l1_ratio	0.01-0.05	0.2721	0.01-0.05	0.9641
shuffle	/	True	/	True
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.8371	0.1-0.9	0.0244
learning_rate	/	adaptive	/	adaptive
eta0	0.001-0.1	0.1632	0.001-0.1	0.012
power_t	0.1-0.5	0.8117	0.1-0.5	0.1533
early_stopping	/	False	/	False
tol	10^{-3}-10^{-2}	0.0077	10^{-3}-10^{-2}	0.0005
random_state	0-10	6	0-10	73
Validation_fraction	0.01-1.0	0.1068	0.01-1.0	0.0633

warm_start	/	False	/	False
average	/	False	/	False
n_iter_no_change	1-10	2	1-10	7

Tablica 7.5.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
verbose	/	False	/	True
normalize	/	False	/	False
precompute	/	auto	/	True
n_nonzero_coefs	100-800	63	1-200	128
eps	10^{-5}-10^{-4}	7.7467e-11	10^{-5}-10^{-4}	4.1976e-05
copy_X	/	True	/	False
fit_path	/	False	/	False
jitter	10^{-5}-10^{-4}	2.7391e-05	10^{-5}-10^{-4}	2.3514e-05
random_state	10-90	55	10-100	75

Tablica 7.5.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-1.0	0.0132	0.0001-0.01	0.0001
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	True	/	True
copy_X	/	False	/	True
max_iter	100-1000	584	100-1000	964
tol	10^{-6}-10^{-3}	4.1665e-05	10^{-6}-10^{-3}	0.0002
warm_start	/	True	/	True
positive	/	False	/	True
random_state	1-100	84	10-100	57

Tablica 7.5.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.1	0.0178	0.01-0.1	0.0542
fit_intercept	/	True	/	True
verbose	/	False	/	False
normalize	/	False	/	False
precompute	/	auto	/	auto
max_iter	100-1000	588	100-1000	237
eps	/	0.0007	/	0.0003
copy_X	/	True	/	True
fit_path	/	True	/	False
positive	/	True	/	False
jitter	0.001-0.1	0.0641	0.01-0.1	0.0983
random_state	10-100	87	1-100	54

Tablica 7.5.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	311	150-350	314
tol	10^{-4}-10^{-3}	0.0007	10^{-7}-10^{-5}	6.6408e-06
alpha_1	10^{-4}-10^{-3}	0.0862	10^{-6}-10^{-5}	7.843e-06
alpha_2	10^{-6}-10^{-5}	9.7103e-05	10^{-8}-10^{-5}	9.4972e-06
lambda_1	10^{-7}-10^{-6}	5.0852e-07	10^{-8}-10^{-6}	2.0002e-07
lambda_2	10^{-6}-10^{-5}	5.1168e-06	10^{-6}-10^{-5}	2.7863e-06
compute_score	/	False	/	True
threshold_lambda	9000-11000	9729.7	9000-10000	9642.2
fit_intercept	/	True	/	True
normalize	/	True	/	False
copy_X	/	True	/	True
verbose	/	False	/	True

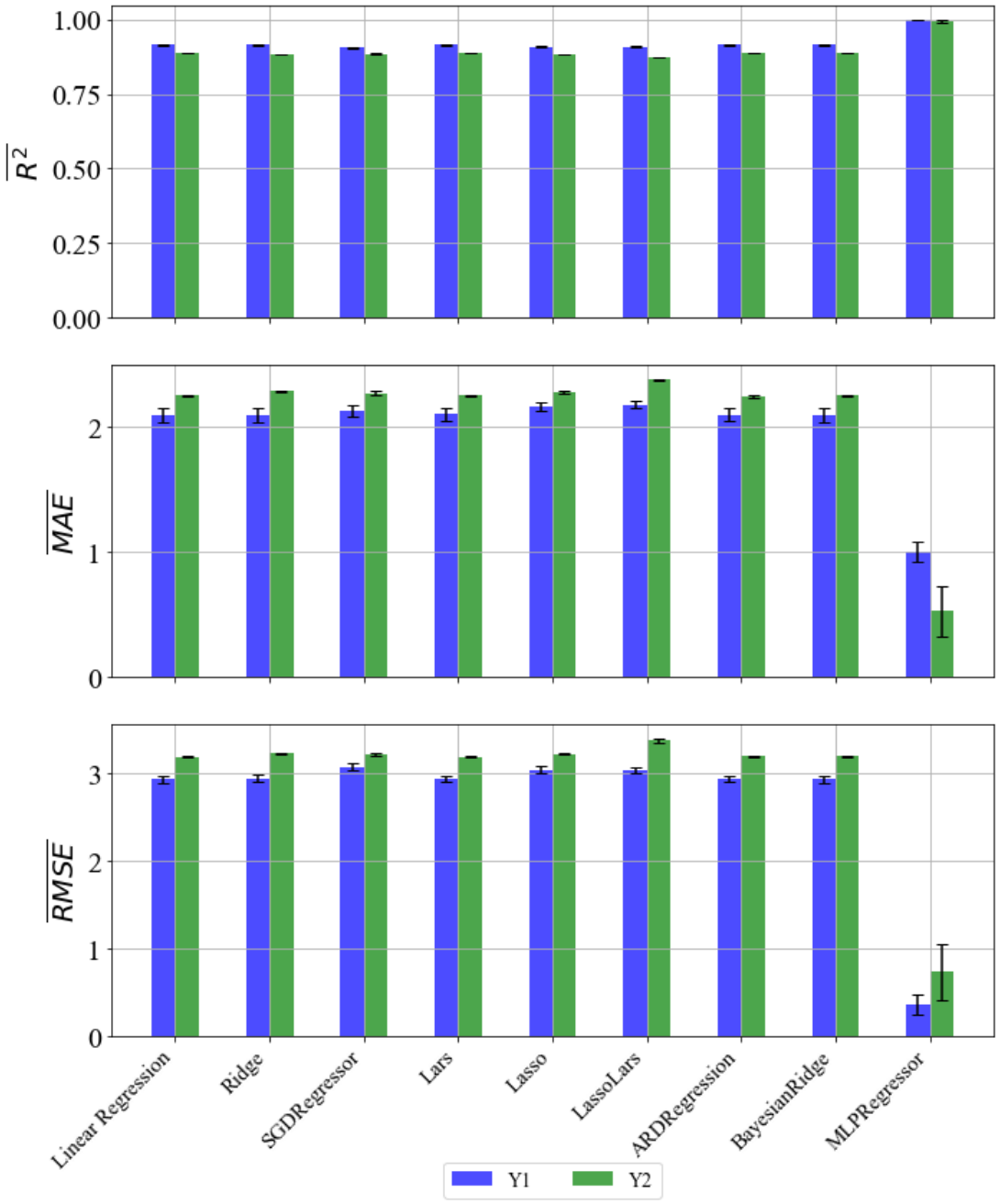
Tablica 7.5.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	303	300-500	436
tol	10^{-3}-10^{-2}	0.0039	10^{-5}-10^{-3}	0.0009
alpha_1	10^{-7}-10^{-5}	1.9232e-07	10^{-7}-10^{-5}	7.0582e-06
alpha_2	10^{-6}-10^{-5}	6.2153e-06	10^{-7}-10^{-5}	5.6613e-06
lambda_1	10^{-6}-10^{-5}	2.6577e-06	10^{-7}-10^{-5}	3.8633e-06
lambda_2	10^{-6}-10^{-5}	2.3367e-06	10^{-7}-10^{-5}	4.2326e-06
alpha_init	10^{-6}-10^{-5}	8.329e-06	10^{-7}-10^{-5}	1.8626e-06
lambda_init	10^{-6}-10^{-5}	5.2656e-06	10^{-7}-10^{-5}	3.1514e-06
compute_score	/	False	/	False
fit_intercept	/	True	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
verbose	/	False	/	True

Tablica 7.5.18. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(76, 39, 66, 94, 75, 32, 67)	10-100	(14, 61, 71, 100)
activation	/	tanh	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0006	10^{-5}-10^{-3}	0.0009
batch_size	do 200	195	do 200	63
learning_rate	/	constant	/	invscaling
learning_rate_init	10^{-5}-10^{-4}	7.4921e-05	10^{-5}-10^{-3}	0.0007
power_t	0.3-0.7	0.4804	0.1-0.9	0.7054
max_iter	300-500	383	200-500	210
shuffle	/	True	/	False
random_state	0-150	14	0-100	90
tol	10^{-5}-10^{-3}	5.6252e-05	10^{-5}-10^{-3}	0.0006
verbose	/	True	/	False
warm_start	/	True	/	True
momentum	0.2-0.8	0.4324	0.1-0.9	0.1977
nesterovs_momentum	/	True	/	True
early_stopping	/	True	/	True
validation_fraction	0.1-0.3	0.1958	0.1-0.3	0.1934
beta_1	0.7-0.9	0.8584	0.7-0.9	0.8628
beta_2	0.8-0.99	0.8477	0.8-0.99	0.9111
epsilon	10^{-9}-10^{-8}	7.3913e-09	10^{-9}-10^{-7}	4.4577e-08
n_iter_no_change	5-20	11	5-20	14
max_fun	10000-20000	14635	10000-20000	17347

7.6. Rezultati dobiveni korištenjem StandardScaler-a



Slika 7.6. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – StandardScaler

Usporedbom rezultata iz poglavlja 6.5. i poglavlja 7.6. vidimo da su rezultati za većinu metoda slični ili neznačajno bolji u slučaju nasumičnog odabira hiperparametara. Najviše se ističu metode Lars i MLPRegressor. Za Lasso i MLPRegressor metode dobiveni su nešto bolji za sve evaluacijske metrike.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 7.6.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	False	/	True
copy_X	/	False	/	True
n_jobs	30-70	52	1-100	54
positive	/	False	/	False

Tablica 7.6.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-10	5.2352	1-10	1.0792
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
max_iter	100-1000	166	100-1000	568
tol	10^{-4}-10^{-2}	0.0047	10^{-4}-10^{-2}	0.0009
solver	/	cholesky	/	cholesky
random_state	50-100	87	40-90	58

Tablica 7.6.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.001-0.03	0.0101	0.001-0.1	0.0282
fit_intercept	/	True	/	True
max_iter	100-1000	258	900-3000	817
penalty	/	l1	/	l1
l1_ratio	0.0-1.0	0.2183	0.2-1.0	0.8152
shuffle	/	False	/	True
verbose	0-0	0	0-0	0
epsilon	0.1-1.0	0.7215	0.01-0.1	0.0288
learning_rate	/	invscaling	/	invscaling
eta0	0.01-0.5	0.4333	0.01-0.1	0.0765
power_t	0.1-1.0	0.8115	0.1-1.0	0.2469
early_stopping	/	True	/	False
tol	10^{-3}-10^{-2}	0.0094	10^{-4}-10^{-3}	0.0002
random_state	1-50	50	0-100	76

Validation_fraction	0.1-1.0	0.4305	0.1-1.0	0.2251
warm_start	/	True	/	True
average	/	True	/	False
n_iter_no_change	1-50	44	1-10	18

Tablica 7.6.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
verbose	/	False	/	False
normalize	/	True	/	False
precompute	/	True	/	auto
n_nonzero_coefs	10-100	51	20-120	113
eps	10^{-12}-10^{-10}	8.2222e-11	10^{-4}-10^{-2}	0.0067
copy_X	/	True	/	False
fit_path	/	True	/	True
jitter	10^{-5}-10^{-4}	3.5403e-05	10^{-5}-10^{-3}	0.0003
random_state	20-100	89	1-50	15

Tablica 7.6.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-1.0	0.0917	0.01-0.1	0.0106
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	True	/	True
copy_X	/	True	/	False
max_iter	3000-10000	4732	4000-9000	4386
tol	10^{-3}-10^{-2}	0.0035	10^{-3}-10^{-2}	0.0061
warm_start	/	False	/	True
positive	/	False	/	False
random_state	20-100	31	10-80	43

Tablica 7.6.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-1.0	0.3596	0.1-1.0	0.1449
fit_intercept	/	True	/	True
verbose	/	False	/	True
normalize	/	False	/	False
precompute	/	True	/	True
max_iter	2000-6000	3688	1500-5000	3302
eps	/	0.0005	/	0.0156
copy_X	/	False	/	True
fit_path	/	False	/	True
positive	/	True	/	True
jitter	0.1-1.0	0.5055	0.01-0.1	0.0543
random_state	350-600	540	500-1000	773

Tablica 7.6.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	30-3000	2652	50-1000	368
tol	10^{-4}-10^{-2}	0.0058	10^{-3}-10^{-2}	0.0074
alpha_1	10^{-6}-10^{-5}	1.7299e-06	10^{-7}-10^{-5}	7.9924e-06
alpha_2	10^{-6}-10^{-5}	8.2291e-06	10^{-6}-10^{-5}	5.2571e-06
lambda_1	10^{-7}-10^{-5}	3.1292e-06	10^{-6}-10^{-5}	1.7199e-06
lambda_2	10^{-7}-10^{-5}	6.5344e-06	10^{-6}-10^{-5}	3.0106e-06
compute_score	/	False	/	True
threshold_lambda	50000-100000	94172.1	50000-90000	62674.6
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	False
verbose	/	False	/	False

Tablica 7.6.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	1000-2500	1734	200-1000	974
tol	10^{-3}-10^{-1}	0.067	10^{-4}-10^{-2}	0.0087
alpha_1	10^{-5}-10^{-4}	7.7981e-05	10^{-6}-10^{-4}	5.7689e-05
alpha_2	10^{-7}-10^{-4}	6.4366e-05	10^{-6}-10^{-4}	7.1954e-05
lambda_1	10^{-5}-10^{-4}	1.3348e-05	10^{-6}-10^{-4}	1.7194e-05
lambda_2	10^{-6}-10^{-4}	1.72e-05	10^{-6}-10^{-4}	8.534e-05
alpha_init	0.01-0.1	0.0662	0.1-1.0	0.9436
lambda_init	0.001-0.1	0.0591	0.001-0.1	0.083
compute_score	/	False	/	True
fit_intercept	/	True	/	True
normalize	/	False	/	True
copy_X	/	True	/	True
verbose	/	True	/	False

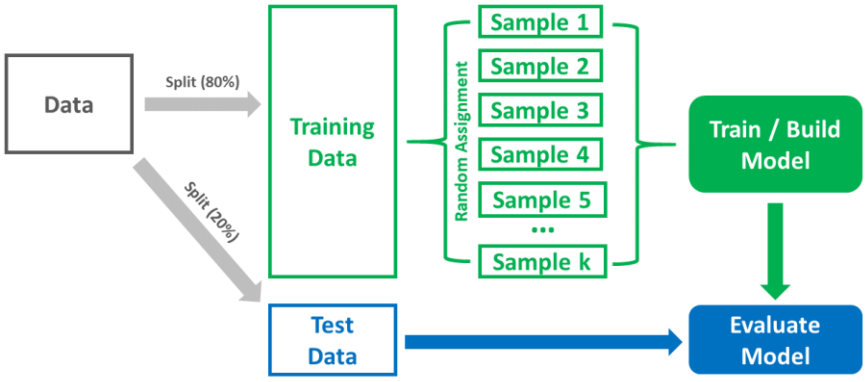
Tablica 7.6.18. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(89, 42, 82, 60, 91, 72, 81)	10-100	(88, 95, 33, 52, 27, 34, 62)
activation	/	relu	/	relu
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0007	10^{-5}-10^{-3}	0.0002
batch_size	do 200	160	do 200	74
learning_rate	/	constant	/	adaptive
learning_rate_init	10^{-5}-10^{-4}	5.8518e-05	10^{-5}-10^{-3}	0.0001
power_t	0.3-0.7	0.6694	0.1-0.9	0.2116
max_iter	300-500	383	200-500	389
shuffle	/	True	/	False
random_state	0-150	113	0-100	41
tol	10^{-5}-10^{-3}	0.0003	10^{-5}-10^{-3}	0.0009
verbose	/	False	/	True
warm_start	/	True	/	True
momentum	0.2-0.8	0.2162	0.1-0.9	0.5597
nesterovs_momentum	/	True	/	False
early_stopping	/	False	/	True
validation_fraction	0.1-0.3	0.2527	0.1-0.3	0.2688
beta_1	0.7-0.9	0.7638	0.7-0.9	0.7314
beta_2	0.8-0.99	0.8506	0.8-0.99	0.8401
epsilon	10^{-9}-10^{-8}	2.9061e-09	10^{-9}-10^{-7}	2.2451e-08
n_iter_no_change	5-20	9	5-20	14
max_fun	10000-20000	11331	10000-20000	18543

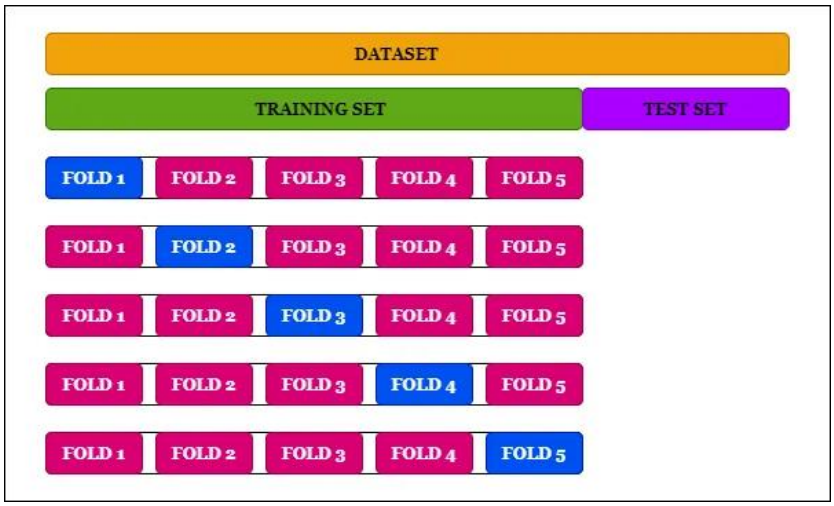
8. UNAKRSNA VALIDACIJA

Kod strojnog učenja najveću važnost pridodajemo sposobnosti dobivenog modela da radi kvalitetne predikcije. Ključni izazov koji se pritom javlja je mogućnost pojave pretreniranja. Vrlo je jednostavno izgraditi model koji je savršeno prilagođen skupu podataka na kojima je treniran, ali daje jako loše rezultate za nove podatke. Jedna od metoda koja se može upotrijebiti za estimaciju performansi izgrađenog modela i za smanjenje rizika od pretreniranja je unakrsna validacija [33]. Jedna indikacija za prepoznavanje pretreniranja je ukoliko su rezultati evaluacijskih metrika znatno lošiji nakon primjene unakrsne validacije nego što su bili ranije.

Općenito za unakrsnu validaciju podatke je potrebno podijeliti u tri skupine. Prva skupina koristi se za treniranje, druga za validaciju, a treća za testiranje. Ovakva podjela se koristi kad postoji dovoljno velika skupina podataka. U slučaju da skupina podataka nije dovoljno velika moguća je pojava problema pri podjeli podataka na dio za treniranje i dio za validaciju. Razlog tome je taj što postoji mogućnost da neke ključne podatkovne točke ne budu uključene u postupak treniranja, pa izgrađeni model ne uspije pravilno naučiti raditi predikcije. Za veće skupine podataka, kao što je to slučaj u ovome radu, takvi problemi se ne javljaju [35]. Postoji nekoliko vrsta unakrsne validacije, neki od kojih su: „Holdout“, „K-Fold“, „Stratified K-Fold“, „Leave-one-out“, itd. U ovom radu koristiti će se „K-Fold“ unakrsna validacija.



Slika 8.1. Shematski prikaz unakrsne validacije [34]



Slika 8.2. Primjer K-Fold unakrsne validacije sa 5 foldova [35]

Na slici 8.2. vidimo primjer „K-Fold“ unakrsne validacije. Čitavi skup podataka se prvobitno dijeli na skup za treniranje i skup za testiranje. Zatim se skup za treniranje dijeli na nekoliko podskupova (eng. folds). „K-Fold“ unakrsna validacija sadrži parametar *k* koji definira u koliko podskupova će se ti podaci podijeliti. Generalno pravilo je uzeti parametar *k* da bude između 5 i 10. Jedan od tih podskupova se koristi za validaciju, a ostali za treniranje modela. Ovaj postupak se ponavlja više puta, svaki put koristeći različitu skupinu podataka kao skup za validaciju. Konačno se izračunavaju srednje vrijednosti rezultata validacije iz svakog koraka kako bi se dobila robusnija procjena performansi modela.

„K-Fold“ metoda unakrsne validacije primijenjena je na skup podataka skupa sa tehnikom nasumičnog pretraživanja hiperparametara kako bi se izgradili čim kvalitetniji modeli, ali istovremeno kako bi se smanjio rizik od pretreniranja.

Kod za ovaj proces je napisan na način da ako su dobivene srednje vrijednosti evaluacijskih metrika ($\overline{R}^2$, $\overline{MAE}$, $\overline{RMSE}$) veće od vrijednosti koje smo zadali tada se proces unakrsne validacije sa nasumičnim pretraživanjem hiperparametara prekida, a ti hiperparametri se koriste za ponovno uvježbavanje modela na skupu podataka za treniranje. U suprotnom se odabiru novi hiperparametri algoritma i proces se ponavlja. Nakon konačnog treniranja izvodi se procjena na testnom skupu podataka kako bi se dobile R^2, MAE i RMSE vrijednosti.

Za unakrsnu validaciju je korištena funkcija *„cross_validate“* iz knjižnice *„sci-kit learn“*.

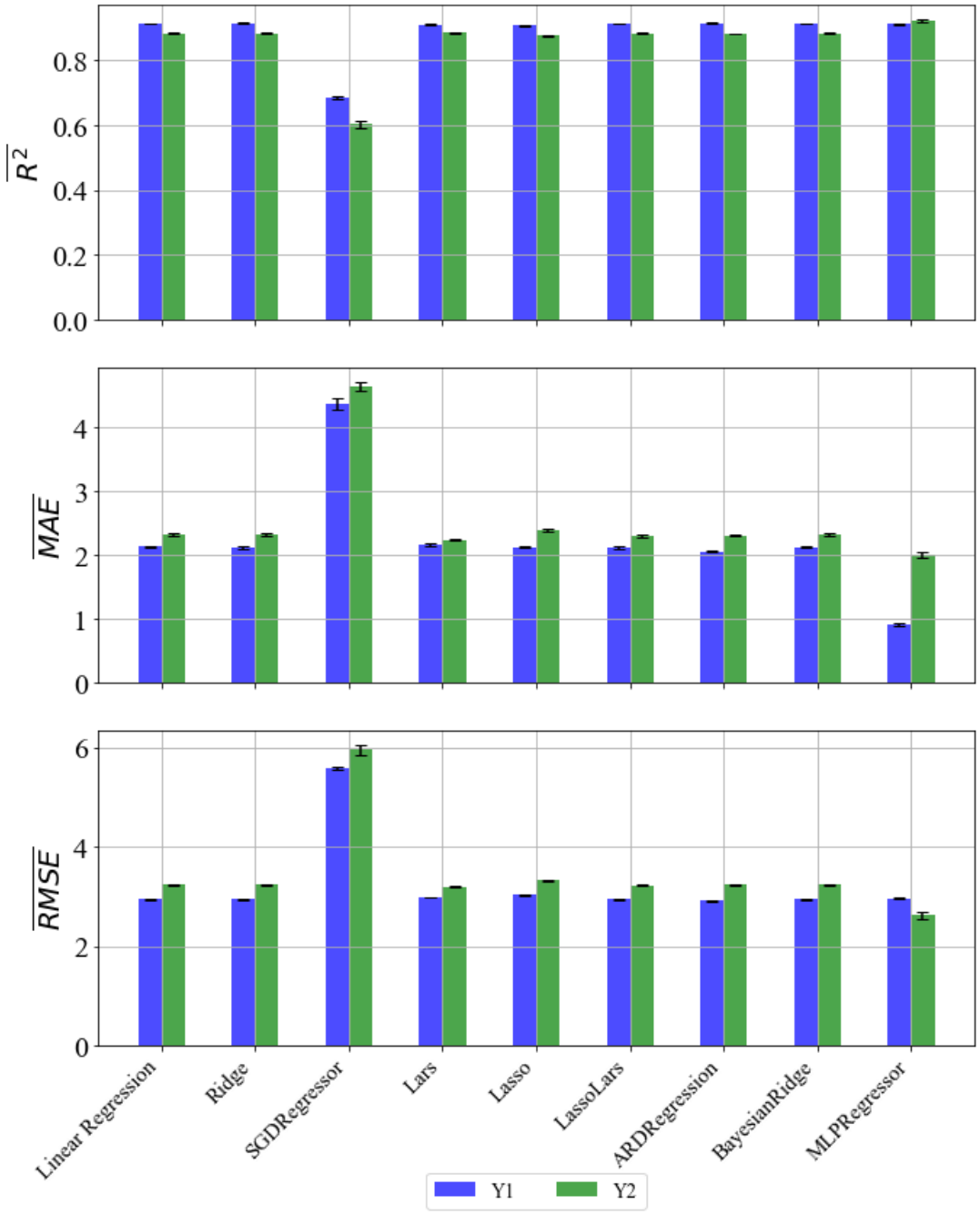
Tablica 8.1. Parametri „cross_validate“ funkcije

Ime parametra	**Tip podataka**	**Ime parametra**	**Tip podataka**
estimator	/	fit_params	Dictionary
groups	/	pre_dispatch	Integer/String
scoring	/	return_train_score	Bool
cv	Integer	reutrn_estimator	Bool
n_jobs	Interger	return_indices	Bool
verbose	Integer	error_score	Float

Parametri *„cross_validate“* funkcije prikazani su u tablici 8.1. Za sve parametre koristiti će se zadane vrijednosti osim za „cv“, „scoring“ i „return_train_score“. Parametrom „cv“ definira se broj podskupova u unakrsnoj validaciji. Parametrom „scoring“ se određuje na koji će se način ocjenjivati predviđanja modela na testnom skupu tijekom postupka unakrsne validacije. Parametar „return_train_score“ određuje hoće li se vraćati rezultati ocjenjivanja za skup za treniranje tijekom izvođenja unakrsne validacije.

U nastavku će biti dane tablice koje sadrže srednje vrijednosti evaluacijskih metrika za sve korištene algoritme nakon nasumičnog pretraživanja hiperparametara i primjene „K-Fold“ unakrsne validacije. Također će se prikazati R^2, MAE i RMSE vrijednosti dobivene nakon konačnog treniranja. Dobiveni rezultati će se usporediti sa onima iz poglavlja 7.

8.1. Rezultati dobiveni bez skaliranja i normalizacije



Slika 8.3. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – bez skaliranja i normalizacije

Usporedbom prethodnih rezultata sa onima iz poglavlja 7.1. vidimo da je „SGDRegressor“ dao puno lošije rezultate za Y1 i Y2 po svim metrikama, „Lasso“ je dao malo lošije rezultate za Y2, a „MLPRegressor“ za Y1. Ostale vrijednosti se nisu značajno promijenile nakon unakrsne validacije.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 8.1.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	False	/	False
normalize	/	False	/	False
copy_X	/	False	/	False
n_jobs	10-85	30	1-100	75
positive	/	False	/	False

Tablica 8.1.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-6	0.0864	0.1-10	0.1756
fit_intercept	/	True	/	False
normalize	/	False	/	True
copy_X	/	True	/	False
max_iter	100-1000	377	100-1000	964
tol	10^{-3}-10^{-2}	0.0077	10^{-4}-10^{-3}	0.0088
solver	/	svd	/	cholesky
random_state	20-100	17	40-100	50

Tablica 8.1.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	huber	/	epsilon_insensitive
alpha	0.001-0.02	0.0793	0.004-0.01	0.0617
fit_intercept	/	True	/	True
max_iter	100-1000	232	100-1000	721
penalty	/	l2	/	l2
l1_ratio	0.2-0.5	0.4671	0.1-0.9	0.0162
shuffle	/	False	/	True
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.8863	0.03-0.1	0.8804
learning_rate	/	adaptive	/	adaptive
eta0	0.01-0.5	0.4269	0.06-0.09	0.3759
power_t	0.2-1.0	0.7284	0.1-0.4	0.6892
early_stopping	/	True	/	False
tol	10^{-3}-10^{-2}	0.0053	10^{-5}-10^{-4}	0.0023
random_state	1-10	1	0-60	2
validation_fraction	0.01-1.0	0.6647	0.01-1.0	0.7563

n_iter_no_change	1-10	5	1-10	9
warm_start	/	False	/	True
average	/	False	/	False

Tablica 8.1.13. Lars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
verbose	/	True	/	False
normalize	/	False	/	True
precompute	/	False	/	True
n_nonzero_coefs	50-90	80	50-90	86
eps	10^{-12}-10^{-10}	1.3986e-11	10^{-5}-10^{-4}	8.9793e-11
copy_X	/	False	/	False
fit_path	/	False	/	False
jitter	10^{-6}-10^{-4}	6.4492e-06	10^{-6}-$2*10^{-5}$	3.3591e-05
random_state	30-80	82	1-50	36

Tablica 8.1.14. Lasso – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.019	0.0373	0.0005-0.008	0.0256
fit_intercept	/	False	/	False
normalize	/	False	/	True
precompute	/	True	/	False
copy_X	/	True	/	False
max_iter	100-1000	506	100-1000	551
tol	10^{-4}-10^{-3}	0.0004	10^{-5}-10^{-4}	0.0003
warm_start	/	False	/	True
positive	/	False	/	False
random_state	30-80	26	40-80	76

Tablica 8.1.15. LassoLars – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.003-0.009	0.0083	0.01-0.1	0.0053
fit_intercept	/	False	/	True
verbose	/	False	/	True
normalize	/	False	/	False
precompute	/	False	/	False
max_iter	100-1000	800	100-1000	312
eps	/	0.0009	/	0.0001
copy_X	/	True	/	True

fit_path	/	False	/	False
positive	/	False	/	False
jitter	0.04-0.1	0.0427	0.04-0.1	0.0794
random_state	70-100	25	30-90	73

Tablica 8.1.16. ARDRegression – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	320	200-350	317
tol	10^{-4}-10^{-3}	0.0009	10^{-6}-10^{-5}	0.0009
alpha_1	10^{-4}-10^{-5}	0.0098	10^{-6}-10^{-5}	0.0064
alpha_2	10^{-6}-10^{-5}	4.5436e-06	10^{-7}-10^{-5}	2.9512e-06
lambda_1	10^{-7}-$4*10^{-7}$	4.1559e-07	10^{-8}-10^{-6}	8.1724e-07
lambda_2	10^{-6}-10^{-5}	9.7233e-06	10^{-6}-10^{-5}	1.5881e-06
compute_score	/	False	/	False
threshold_lambda	9800-10400	9451.2	9800-10500	10408.5
fit_intercept	/	True	/	False
normalize	/	False	/	True
copy_X	/	False	/	False
verbose	/	True	/	False

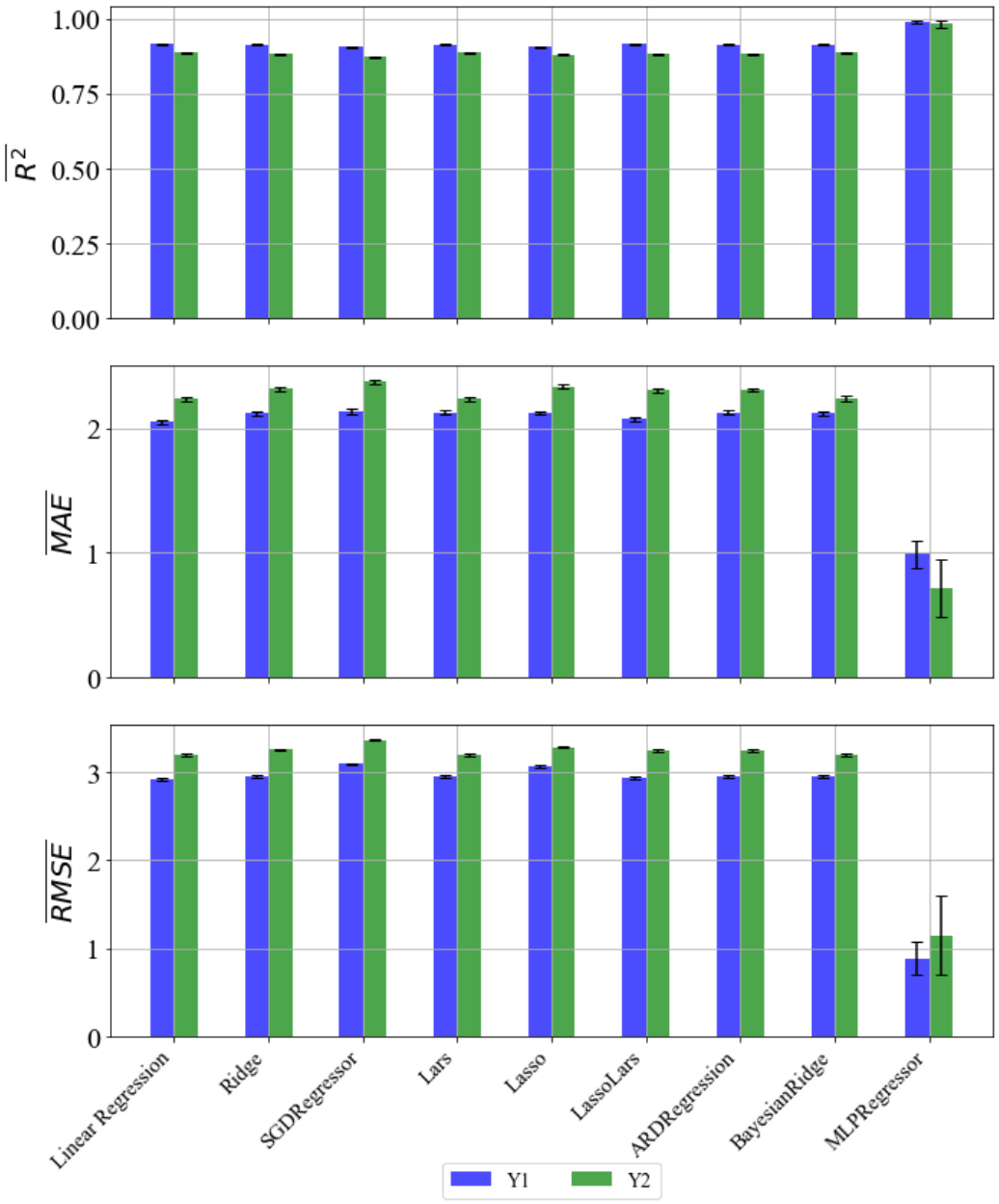
Tablica 8.1.17. BayesianRidge – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	338	300-500	367
tol	$2*10^{-3}$-10^{-3}	0.0064	$2*10^{-4}$-10^{-3}	0.0034
alpha_1	10^{-6}-10^{-5}	6.3313e-06	10^{-6}-$8*10^{-6}$	1.4657e-06
alpha_2	10^{-7}-10^{-5}	6.6355e-06	10^{-7}-10^{-5}	4.3389e-06
lambda_1	10^{-6}-10^{-5}	2.5474e-06	10^{-6}-$8*10^{-6}$	6.3173e-06
lambda_2	10^{-7}-10^{-5}	5.6757e-06	$5*10^{-7}$-10^{-5}	7.0771e-06
alpha_init	10^{-7}-10^{-4}	9.4106e-07	10^{-7}-$5*10^{-6}$	3.8302e-06
lambda_init	10^{-7}-10^{-5}	8.8282e-06	10^{-6}-10^{-5}	9.6761e-06
compute_score	/	True	/	True
fit_intercept	/	False	/	False
normalize	/	False	/	False
copy_X	/	False	/	False
verbose	/	False	/	False

Tablica 8.1.18. MLPRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	[39]	10-100	[51, 87, 70, 81, 49, 95, 72, 58]
activation	/	identity	/	relu
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	9.5242e-05	10^{-5}-10^{-3}	0.0007
batch_size	do 200	170	do 200	176
learning_rate	/	invscaling	/	constant
learning_rate_init	10^{-5}-10^{-4}	3.4309e-05	10^{-5}-10^{-3}	4.2496e-05
power_t	0.3-0.7	0.6559	0.1-0.9	0.3813
max_iter	300-500	460	200-500	452
shuffle	/	True	/	False
random_state	0-150	83	0-100	147
tol	10^{-5}-10^{-3}	0.0009	10^{-5}-10^{-3}	0.0001
verbose	/	False	/	False
warm_start	/	True	/	False
momentum	0.2-0.8	0.7932	0.1-0.9	0.7182
nesterovs_momentum	/	True	/	False
early_stopping	/	True	/	True
validation_fraction	0.1-0.3	0.1904	0.1-0.3	0.2656
beta_1	0.7-0.9	0.8498	0.7-0.9	0.737
beta_2	0.8-0.99	0.8925	0.8-0.99	0.8423
epsilon	10^{-9}-10^{-8}	8.3876e-09	10^{-9}-10^{-7}	2.1847e-09
n_iter_no_change	5-20	19	5-20	7
max_fun	10000-20000	10798	10000-20000	12287

8.2. Rezultati dobiveni korištenjem MaxAbsScaler-a



Slika 8.4. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – MaxAbsScaler

Usporedbom prethodnih rezultata sa onima iz poglavlja 7.2. vidimo da je „SGDRegressor“ dao malo lošije rezultate za Y1 i Y2 po svim metrikama, a „Lasso“ je dao malo lošije rezultate za Y1. Ostale vrijednosti se nisu značajno promijenile nakon unakrsne validacije..

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 8.2.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	True	/	False
copy_X	/	True	/	True
n_jobs	0-10	49	1-100	100
positive	/	False	/	False

Tablica 8.2.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-1.5	0.0487	0.08-0.1	0.2168
fit_intercept	/	False	/	False
normalize	/	True	/	False
copy_X	/	True	/	False
max_iter	10-100	156	100-1000	722
tol	10^{-6}-10^{-3}	0.002	10^{-7}-10^{-6}	0.0036
solver	/	lsqr	/	cholesky
random_state	0-10	87	50-100	78

Tablica 8.2.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	epsilon_insensitive	/	squared_loss
alpha	0.001-0.001	0.0027	0.001-0.005	0.003
fit_intercept	/	True	/	True
max_iter	100-1000	335	100-1000	943
penalty	/	l1	/	l2
l1_ratio	0.0-0.3	0.1536	0.2-0.9	0.0059
shuffle	/	False	/	True
verbose	0-0	0	0-0	0
epsilon	0.2-1.0	0.8608	0.03-0.08	0.8438
learning_rate	/	adaptive	/	adaptive
eta0	0.15-0.35	0.4413	0.05-0.1	0.3091
power_t	0.1-0.7	0.2173	0.1-0.5	0.1892
early_stopping	/	True	/	False
tol	10^{-3}-10^{-2}	0.0075	10^{-4}-10^{-3}	0.0023
random_state	3-7	10	10-60	8
validation_fraction	0.01-1.0	0.1876	0.01-1.0	0.208
n_iter_no_change	1-10	10	1-10	9
warm_start	/	False	/	False
average	/	False	/	True

Tablica 8.2.13. Lars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	False	/	True
verbose	/	True	/	True
normalize	/	True	/	True
precompute	/	Auto	/	Auto
n_nonzero_coefs	13-85	14	5-15	45
eps	10^{-12}-10^{-10}	9.3137e-11	10^{-6}-10^{-4}	1.7225e-12
copy_X	/	False	/	True
fit_path	/	True	/	True
jitter	10^{-7}-10^{-4}	1.3511e-05	10^{-5}-10^{-4}	8.306e-05
random_state	10-100	71	3-35	80

Tablica 8.2.14. Lasso – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.008-0.02	0.0223	0.0001-0.001	0.0178
fit_intercept	/	False	/	True
normalize	/	False	/	False
precompute	/	False	/	False
copy_X	/	True	/	False
max_iter	100-1000	210	100-1000	490
tol	10^{-4}-10^{-3}	0.0007	10^{-4}-10^{-3}	9.121e-05
warm_start	/	True	/	True
positive	/	False	/	False
random_state	20-82	62	10-80	69

Tablica 8.2.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.09	0.003	0.01-0.2	0.0049
fit_intercept	/	True	/	False
verbose	/	True	/	True
normalize	/	False	/	False
precompute	/	False	/	Auto
max_iter	100-1000	813	100-1000	409
eps	/	0.0009	/	0.0003
copy_X	/	True	/	False
fit_path	/	False	/	True
positive	/	False	/	False
jitter	0.02-0.1	0.0482	0.05-0.1	0.0176
random_state	40-100	38	30-70	52

Tablica 8.2.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	260-350	298	150-300	281
tol	10^{-5}-10^{-3}	0.0002	10^{-6}-10^{-5}	0.0006
alpha_1	10^{-2}-10^{-4}	0.0039	10^{-6}-10^{-5}	0.0038
alpha_2	10^{-7}-10^{-5}	3.5994e-07	10^{-6}-10^{-5}	9.3772e-06
lambda_1	10^{-7}-10^{-6}	4.0301e-07	10^{-7}-10^{-6}	9.4639e-07
lambda_2	10^{-6}-10^{-5}	2.0503e-06	10^{-6}-10^{-5}	8.0488e-06
compute_score	/	False	/	True
threshold_lambda	9300-10500	9457.6	9000-11000	9180.6
fit_intercept	/	False	/	False
normalize	/	True	/	True
copy_X	/	False	/	False
verbose	/	False	/	True

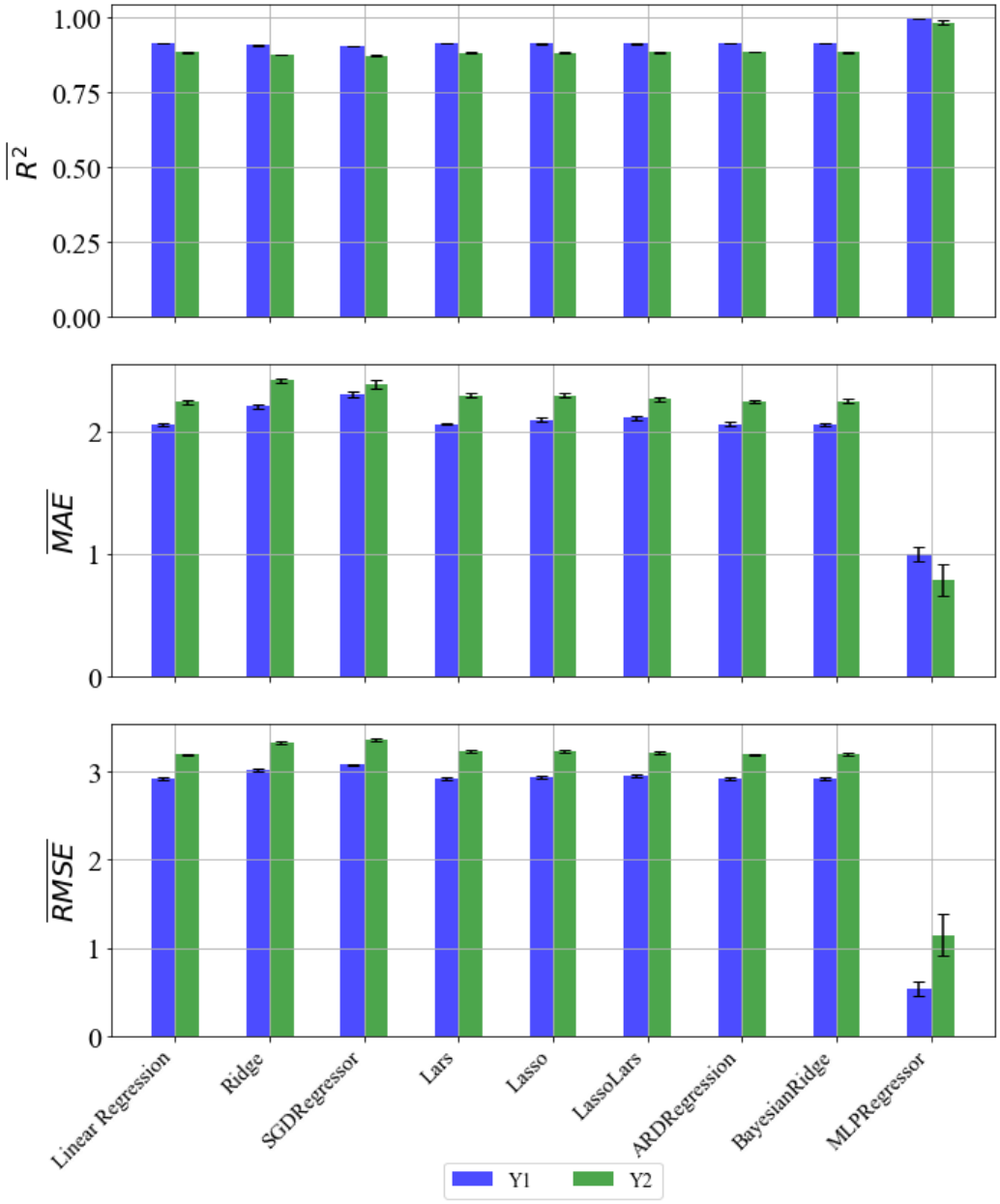
Tablica 8.2.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	382	300-500	376
tol	10^{-}-10^{-2}	0.0094	10^{-4}-10^{-3}	0.0089
alpha_1	10^{-6}-$7*10^{-6}$	7.5266e-06	10^{-7}-10^{-5}	1.5651e-06
alpha_2	$2*10^{-6}$-$7*10^{-6}$	2.9376e-06	10^{-7}-10^{-5}	9.4062e-06
lambda_1	$2*10^{-6}$-$8*10^{-6}$	3.3171e-06	10^{-7}-10^{-5}	2.7125e-06
lambda_2	$4*10^{-7}$-10^{-5}	9.4916e-07	10^{-7}-10^{-5}	2.5157e-06
alpha_init	$2*10^{-7}$-10^{-6}	1.7422e-07	10^{-7}-10^{-5}	1.4052e-06
lambda_init	10^{-7}-$2*10^{-6}$	3.8010e-06	10^{-7}-10^{-5}	4.2944e-06
compute_score	/	False	/	False
fit_intercept	/	False	/	True
normalize	/	True	/	True
copy_X	/	True	/	True
verbose	/	False	/	False

Tablica 8.2.18. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	[90, 13]	10-100	[44, 78]
activation	/	tanh	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0004	10^{-5}-10^{-3}	1.9209e-05
batch_size	do 200	131	do 200	171
learning_rate	/	invscaling	/	invscaling
learning_rate_init	10^{-5}-10^{-4}	3.0044e-05	10^{-5}-10^{-3}	6.3339e-05
power_t	0.3-0.7	0.5104	0.1-0.9	0.4831
max_iter	300-500	491	200-500	470
shuffle	/	False	/	True
random_state	0-150	39	0-100	25
tol	10^{-5}-10^{-3}	0.0005	10^{-5}-10^{-3}	0.0003
verbose	/	False	/	False
warm_start	/	True	/	True
momentum	0.2-0.8	0.7005	0.1-0.9	0.5401
nesterovs_momentum	/	False	/	True
early_stopping	/	True	/	False
validation_fraction	0.1-0.3	0.1888	0.1-0.3	0.1206
beta_1	0.7-0.9	0.8429	0.7-0.9	0.7585
beta_2	0.8-0.99	0.8257	0.8-0.99	0.8624
epsilon	10^{-9}-10^{-8}	8.9646e-09	10^{-9}-10^{-7}	4.8221e-09
n_iter_no_change	5-20	5	5-20	13
max_fun	10000-20000	16963	10000-20000	15441

8.3. Rezultati dobiveni korištenjem MinMaxScaler-a



Slika 8.5. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – MinMaxScaler

Usporedbom prethodnih rezultata sa onima iz poglavlja 7.3. vidimo da je „SGDRegressor“ dao malo lošije rezultate za Y1 i Y2 po svim metrikama. Ostale vrijednosti se nisu značajno promijenile nakon unakrsne validacije.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 8.3.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	False	/	False
normalize	/	False	/	False
copy_X	/	False	/	True
n_jobs	10-85	13	1-100	48
positive	/	False	/	False

Tablica 8.3.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-6	2.1088	0.1-10	2.2247
fit_intercept	/	False	/	False
normalize	/	True	/	True
copy_X	/	False	/	True
max_iter	100-1000	177	100-1000	283
tol	10^{-3}-10^{-2}	0.0012	10^{-4}-10^{-3}	0.0085
solver	/	sparse_cg	/	cholesky
random_state	20-100	97	40-100	69

Tablica 8.3.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	epsilon_insensitive	/	epsilon_insensitive
alpha	0.001-0.02	0.0678	0.004-0.01	0.0112
fit_intercept	/	False	/	True
max_iter	100-1000	346	100-1000	670
penalty	/	l1	/	l1
l1_ratio	0.2-0.5	0.1284	0.1-0.9	0.0208
shuffle	/	False	/	False
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.6341	0.03-0.1	0.6967
learning_rate	/	constant	/	adaptive
eta0	0.01-0.5	0.0771	0.06-0.09	0.4743
power_t	0.2-1.0	0.5288	0.1-0.4	0.1659
early_stopping	/	False	/	True
tol	10^{-3}-10^{-2}	0.0011	10^{-5}-10^{-4}	0.0011
random_state	1-10	10	0-60	6
validation_fraction	0.01-1.0	0.6668	0.01-1.0	0.3119
n_iter_no_change	1-10	9	1-10	3
warm_start	/	True	/	False
average	/	False	/	False

Tablica 8.3.13. Lars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
verbose	/	True	/	True
normalize	/	False	/	False
precompute	/	Auto	/	False
n_nonzero_coefs	50-90	61	5-9	71
eps	10^{-12}-10^{-10}	9.8413e-11	10^{-5}-10^{-4}	1.84951060723e-11
copy_X	/	True	/	True
fit_path	/	True	/	False
jitter	10^{-6}-10^{-4}	1.1011e-05	10^{-6}-2*10^{-5}	8.1218e-05
random_state	30-80	30	1-50	54

Tablica 8.3.14. Lasso – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.019	0.01	0.0005-0.008	0.0147
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	True	/	True
copy_X	/	True	/	True
max_iter	100-1000	684	100-1000	342
tol	10^{-4}-10^{-3}	0.0004	10^{-5}-10^{-4}	0.0009
warm_start	/	True	/	True
positive	/	False	/	False
random_state	30-80	33	40-80	10

Tablica 8.3.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.003-0.009	0.0104	0.01-0.1	0.0045
fit_intercept	/	True	/	True
verbose	/	False	/	False
normalize	/	False	/	False
precompute	/	Auto	/	False
max_iter	100-1000	227	100-1000	511
eps	/	0.0009	/	0.0008
copy_X	/	True	/	True
fit_path	/	True	/	False
positive	/	False	/	False
jitter	0.04-0.1	0.0965	0.04-0.1	0.0372
random_state	70-100	59	30-90	46

Tablica 8.3.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	348	200-350	318
tol	10^{-4}-10^{-3}	0.0003	10^{-6}-10^{-5}	0.0001
alpha_1	10^{-4}-10^{-5}	0.0031	10^{-6}-10^{-5}	0.0028
alpha_2	10^{-6}-10^{-5}	1.1574e-06	10^{-7}-10^{-5}	1.6311e-06
lambda_1	10^{-7}-$4*10^{-7}$	2.68e-07	10^{-8}-10^{-6}	6.7193e-07
lambda_2	10^{-6}-10^{-5}	3.5146e-06	10^{-6}-10^{-5}	3.8147e-06
compute_score	/	False	/	True
threshold_lambda	9800-10400	10882	9800-10500	10633.6
fit_intercept	/	True	/	True
normalize	/	True	/	False
copy_X	/	True	/	True
verbose	/	False	/	False

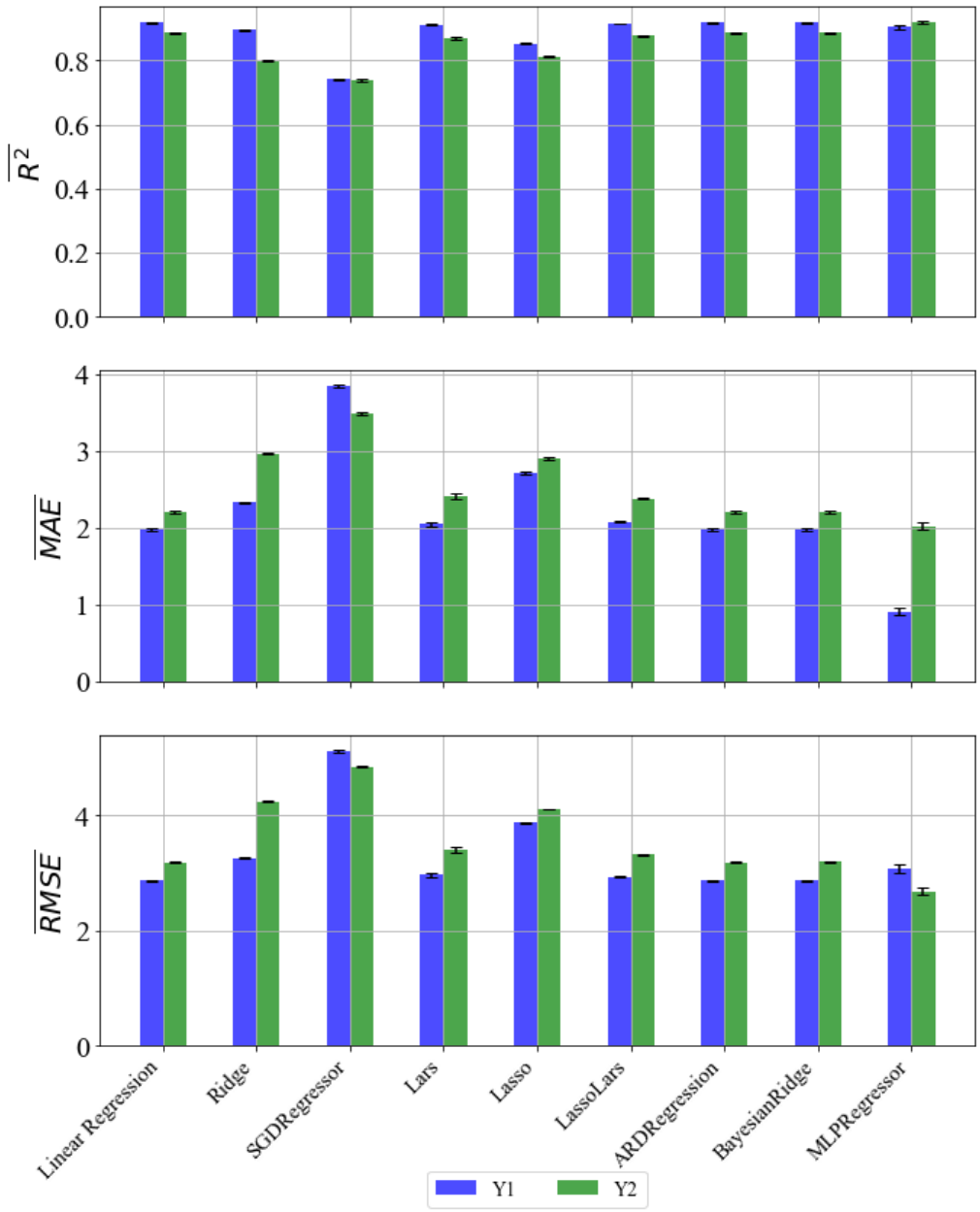
Tablica 8.3.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	313	300-500	324
tol	$2*10^{-3}$-10^{-3}	0.0011	$2*10^{-4}$-10^{-3}	0.0004
alpha_1	10^{-6}-10^{-5}	1.7838e-06	10^{-6}-$8*10^{-6}$	2.0648e-06
alpha_2	10^{-7}-10^{-5}	2.4664e-06	10^{-7}-10^{-5}	1.3599e-06
lambda_1	10^{-6}-10^{-5}	7.1437e-06	10^{-6}-$8*10^{-6}$	7.2021e-06
lambda_2	10^{-7}-10^{-5}	3.0833e-06	$5*10^{-7}$-10^{-5}	8.7498e-06
alpha_init	10^{-7}-10^{-4}	2.8702e-06	10^{-7}-$5*10^{-6}$	2.0111e-06
lambda_init	10^{-7}-10^{-5}	5.4209e-06	10^{-6}-10^{-5}	1.9504e-06
compute_score	/	False	/	True
fit_intercept	/	True	/	True
normalize	/	True	/	False
copy_X	/	True	/	True
verbose	/	False	/	False

Tablica 8.3.18. MLPRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	[34, 54, 38, 36, 18]	10-100	[93, 36, 88]
activation	/	tanh	/	Relu
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0007	10^{-5}-10^{-3}	0.0006
batch_size	do 200	141	do 200	142
learning_rate	/	adaptive	/	adaptive
learning_rate_init	10^{-5}-10^{-4}	7.1083e-05	10^{-5}-10^{-3}	3.0262e-05
power_t	0.3-0.7	0.4635	0.1-0.9	0.3695
max_iter	300-500	481	200-500	304
shuffle	/	False	/	True
random_state	0-150	90	0-100	79
tol	10^{-5}-10^{-3}	0.0009	10^{-5}-10^{-3}	0.0003
verbose	/	False	/	False
warm_start	/	True	/	False
momentum	0.2-0.8	0.4754	0.1-0.9	0.7043
nesterovs_momentum	/	True	/	False
early_stopping	/	False	/	True
validation_fraction	0.1-0.3	0.2355	0.1-0.3	0.2057
beta_1	0.7-0.9	0.7998	0.7-0.9	0.8626
beta_2	0.8-0.99	0.909	0.8-0.99	0.8565
epsilon	10^{-9}-10^{-8}	1.2806e-09	10^{-9}-10^{-7}	4.7584e-09
n_iter_no_change	5-20	17	5-20	13
max_fun	10000-20000	11455	10000-20000	17848

8.4. Rezultati dobiveni korištenjem Normalizer-a



Slika 8.6. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – Normalizer

Usporedbom prethodnih rezultata sa onima iz poglavlja 7.4. vidimo da je „Lasso“ dao malo lošije rezultate za Y1 i Y2 po svim metrikama, a „Ridge“ malo bolje rezultate za Y1 i Y2. Ostale vrijednosti se nisu značajno promijenile nakon unakrsne validacije.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 8.4.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	true	/	true
normalize	/	true	/	true
copy_X	/	true	/	true
n_jobs	10-50	30	1-100	7
positive	/	false	/	false

Tablica 8.4.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-0.7	0.0412	1.0-1.04	0.5272
fit_intercept	/	true	/	true
normalize	/	true	/	true
copy_X	/	true	/	true
max_iter	100-1000	493	100-1000	292
tol	10^{-3}-10^{-2}	0.0032	10^{-4}-5*10^{-4}	0.0082
solver	/	saga	/	svd
random_state	5-30	24	80-100	21

Tablica 8.4.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.00001-0.1	0.0301	0.00001-0.1	0.0012
fit_intercept	/	true	/	false
max_iter	900-3000	675	900-3000	934
penalty	/	L1	/	l1
l1_ratio	0.01-0.05	0.1596	0.01-0.05	0.4537
shuffle	/	false	/	false
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.2023	0.1-0.9	0.2716
learning_rate	/	adaptive	/	adaptive
eta0	0.001-0.1	0.1195	0.001-0.1	0.4001
power_t	0.1-0.5	0.3229	0.1-0.5	0.5404
early_stopping	/	false	/	false
tol	10^{-3}-10^{-2}	0.009	10^{-3}-10^{-2}	0.0011
random_state	0-10	1	0-10	6
Validation_fraction	0.01-1.0	0.5579	0.01-1.0	0.3667
warm_start	/	false	/	False
average	/	false	/	false
n_iter_no_change	1-10	9	1-10	7

Tablica 8.4.13. Lars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	true	/	true
verbose	/	false	/	true
normalize	/	false	/	false
precompute	/	auto	/	auto
n_nonzero_coefs	100-800	49	80-120	23
eps	10^{-5}-10^{-4}	6.2072e-11	10^{-10}-10^{-3}	9.8427e-11
copy_X	/	true	/	true
fit_path	/	false	/	false
jitter	10^{-5}-10^{-4}	3.3364e-05	10^{-5}-10^{-3}	1.2849e-05
random_state	10-90	12	70-90	42

Tablica 8.4.14. Lasso – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.03	0.0201	0.0004-0.003	0.032
fit_intercept	/	true	/	true
normalize	/	true	/	true
precompute	/	true	/	false
copy_X	/	false	/	true
max_iter	100-1000	319	400-1000	155
tol	10^{-6}-10^{-3}	0.0002	10^{-5}-10^{-3}	0.0004
warm_start	/	false	/	true
positive	/	true	/	true
random_state	20-90	67	10-100	6

Tablica 8.4.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.00-0.01	0.0016	0.005-0.01	0.006
fit_intercept	/	true	/	true
verbose	/	true	/	false
normalize	/	true	/	true
precompute	/	true	/	true
max_iter	100-1000	511	100-1000	457
eps	/	0.0008	/	0.0003
copy_X	/	true	/	true
fit_path	/	true	/	false
positive	/	false	/	false
jitter	0.01-0.1	0.0359	0.01-0.1	0.0914
random_state	1-100	94	50-100	88

Tablica 8.4.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	330	150-350	309
tol	10^{-4}-10^{-3}	0.0001	10^{-6}-10^{-5}	2.9213e-05
alpha_1	10^{-4}-10^{-3}	0.0057	10^{-6}-10^{-5}	0.0071
alpha_2	10^{-6}-10^{-5}	9.9783e-06	10^{-7}-10^{-6}	1.0309e-06
lambda_1	10^{-7}-10^{-6}	9.6229e-07	10^{-7}-10^{-6}	9.3172e-07
lambda_2	10^{-6}-10^{-5}	5.1401e-06	10^{-6}-10^{-5}	2.6956e-06
compute_score	/	true	/	false
threshold_lambda	9000-11000	10034.4	9000-10000	10807.9
fit_intercept	/	true	/	true
normalize	/	false	/	true
copy_X	/	true	/	true
verbose	/	true	/	true

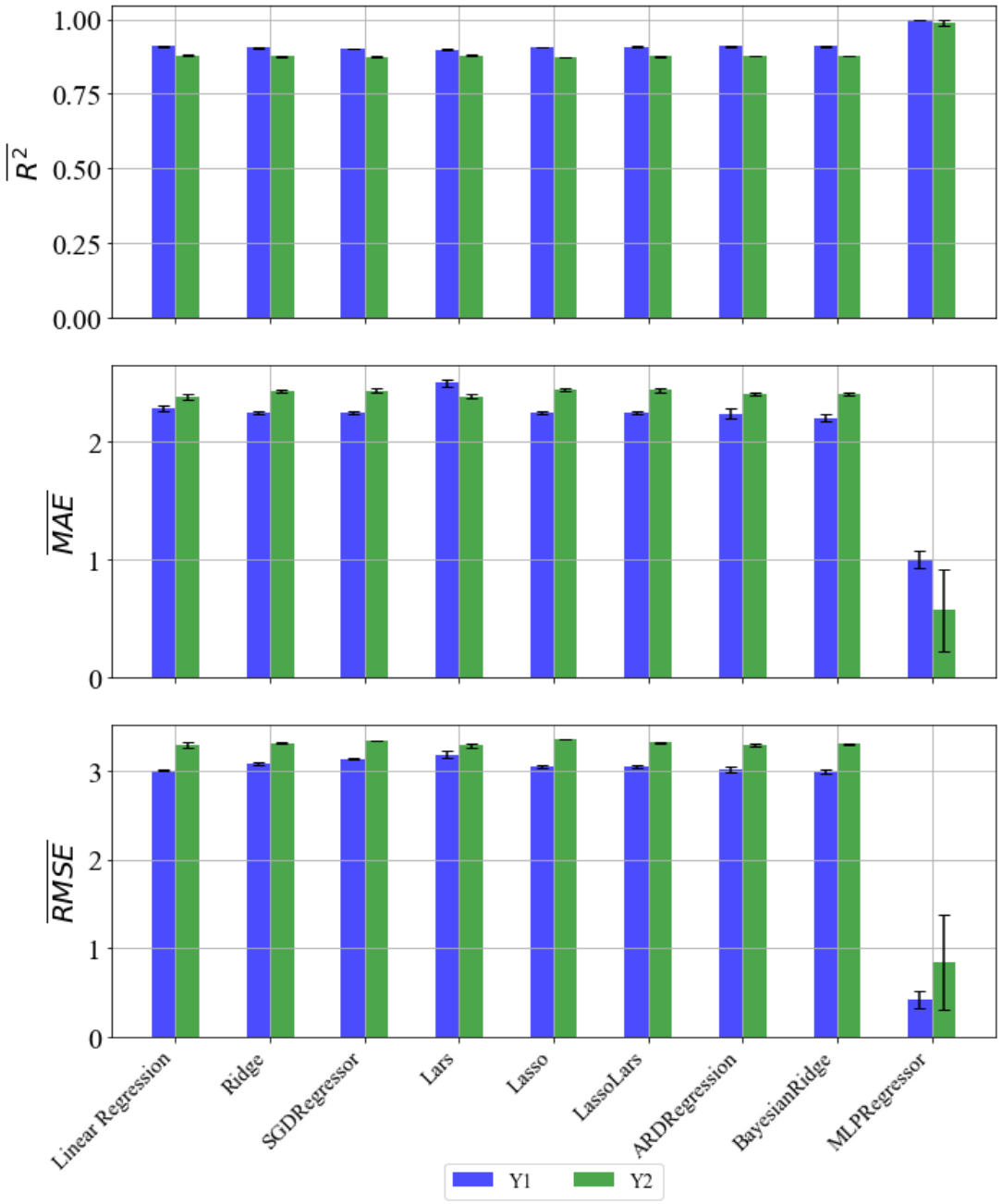
Tablica 8.4.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	342	300-500	326
tol	10^{-4}-10^{-2}	0.004	10^{-4}-10^{-3}	0.0078
alpha_1	10^{-7}-10^{-5}	1.3422e-06	10^{-6}-10^{-5}	6.1454e-06
alpha_2	10^{-7}-10^{-5}	9.4161e-06	$2*10^{-6}$-10^{-5}	3.5019e-06
lambda_1	10^{-7}-10^{-5}	8.1371e-06	$5*10^{-6}$-10^{-5}	3.1362e-06
lambda_2	10^{-7}-10^{-5}	6.2739e-06	$8*10^{-6}$-10^{-5}	8.1979e-06
alpha_init	10^{-7}-10^{-5}	5.0969e-06	10^{-7}-10^{-5}	3.2511e-06
lambda_init	10^{-7}-10^{-5}	7.4252e-07	$2*10^{-6}$-10^{-5}	4.8771e-06
compute_score	/	true	/	false
fit_intercept	/	true	/	true
normalize	/	false	/	true
copy_X	/	true	/	true
verbose	/	false	/	true

Tablica 8.4.18. MLPRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	[26, 63, 94, 52, 25, 21]	10-100	[45, 95, 63, 56, 68, 77]
activation	/	tanh	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0008	10^{-5}-10^{-3}	0.0007
batch_size	do 200	77	do 200	54
learning_rate	/	adaptive	/	constant
learning_rate_init	10^{-5}-10^{-4}	8.8482e-05	10^{-5}-10^{-3}	1.7511e-05
power_t	0.3-0.7	0.6878	0.1-0.9	0.5893
max_iter	300-500	325	200-500	407
shuffle	/	true	/	false
random_state	0-150	14	0-100	118
tol	10^{-5}-10^{-3}	0.0006	10^{-5}-10^{-3}	0.0007
verbose	/	true	/	true
warm_start	/	false	/	false
momentum	0.2-0.8	0.2541	0.1-0.9	0.3047
nesterovs_momentum	/	true	/	false
early_stopping	/	true	/	false
validation_fraction	0.1-0.3	0.223	0.1-0.3	0.2266
beta_1	0.7-0.9	0.8222	0.7-0.9	0.7415
beta_2	0.8-0.99	0.8056	0.8-0.99	0.9837
epsilon	10^{-9}-10^{-8}	2.0761e-09	10^{-9}-10^{-7}	4.3089e-09
n_iter_no_change	5-20	6	5-20	8
max_fun	10000-20000	16302	10000-20000	10091

8.5. Rezultati dobiveni korištenjem PowerTransformer-a



Slika 8.7. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – PowerTransformer

Usporedbom prethodnih rezultata sa onima iz poglavlja 7.5. vidimo da se vrijednosti nisu značajno promijenile nakon unakrsne validacije.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 8.5.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	False	/	True
n_jobs	1-100	34	1-100	29
positive	/	True	/	false

Tablica 8.5.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-0.7	9.1895	2.0-5.0	2.5072
fit_intercept	/	true	/	true
normalize	/	false	/	false
copy_X	/	true	/	false
max_iter	100-1000	510	100-1000	197
tol	10^{-3}-10^{-2}	0.0067	10^{-4}-10^{-3}	0.0024
solver	/	svd	/	cholesky
random_state	5-30	21	30-80	27

Tablica 8.5.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	epsilon_insensitive	/	squared_loss
alpha	0.00001-0.1	0.0076	0.00001-0.1	0.0489
fit_intercept	/	true	/	true
max_iter	900-3000	687	900-3000	471
penalty	/	L1	/	L1
l1_ratio	0.01-0.05	0.4315	0.01-0.05	0.0375
shuffle	/	false	/	false
verbose	0-0	0	0-0	0
epsilon	0.1-0.9	0.9057	0.1-0.9	0.3662
learning_rate	/	adaptive	/	adaptive
eta0	0.001-0.1	0.375	0.001-0.1	0.1505
power_t	0.1-0.5	0.1806	0.1-0.5	0.3983
early_stopping	/	true	/	true
tol	10^{-3}-10^{-2}	0.005	10^{-3}-10^{-2}	0.0086
random_state	0-10	6	0-10	8
Validation_fraction	0.01-1.0	0.3752	0.01-1.0	0.7344
warm_start	/	true	/	False
average	/	false	/	false
n_iter_no_change	1-10	2	1-10	7

Tablica 8.5.13. Lars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	true	/	true
verbose	/	false	/	true
normalize	/	true	/	true
precompute	/	false	/	auto
n_nonzero_coefs	100-800	33	1-200	100
eps	10^{-5}-10^{-4}	7.5382e-11	10^{-5}-10^{-4}	8.5878e-11
copy_X	/	true	/	true
fit_path	/	false	/	true
jitter	10^{-5}-10^{-4}	4.2351e-05	10^{-5}-10^{-4}	2.4232e-05
random_state	10-90	19	10-100	68

Tablica 8.5.14. Lasso – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-1.0	0.1239	0.0001-0.01	0.3337
fit_intercept	/	true	/	true
normalize	/	false	/	false
precompute	/	true	/	true
copy_X	/	false	/	false
max_iter	100-1000	193	100-1000	951
tol	10^{-6}-10^{-3}	0.0009	10^{-6}-10^{-3}	0.0005
warm_start	/	true	/	true
positive	/	true	/	true
random_state	1-100	87	10-100	70

Tablica 8.5.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-0.1	0.0998	0.01-0.1	0.0604
fit_intercept	/	true	/	true
verbose	/	true	/	false
normalize	/	false	/	false
precompute	/	true	/	true
max_iter	100-1000	407	100-1000	816
eps	/	0.0009	/	0.0001
copy_X	/	true	/	true
fit_path	/	false	/	true
positive	/	true	/	false
jitter	0.001-0.1	0.0796	0.01-0.1	0.0622
random_state	10-100	59	1-100	42

Tablica 8.5.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	250-350	338	150-350	291
tol	10^{-4}-10^{-3}	0.0009	10^{-7}-10^{-5}	0.0001
alpha_1	10^{-4}-10^{-3}	0.0048	10^{-6}-10^{-5}	0.0038
alpha_2	10^{-6}-10^{-5}	8.2921e-06	10^{-8}-10^{-5}	4.9242e-06
lambda_1	10^{-7}-10^{-6}	1.9082e-07	10^{-8}-10^{-6}	8.0255e-07
lambda_2	10^{-6}-10^{-5}	6.2402e-06	10^{-6}-10^{-5}	8.3753e-06
compute_score	/	false	/	false
threshold_lambda	9000-11000	10680	9000-10000	9746.6
fit_intercept	/	True	/	true
normalize	/	true	/	false
copy_X	/	True	/	true
verbose	/	True	/	false

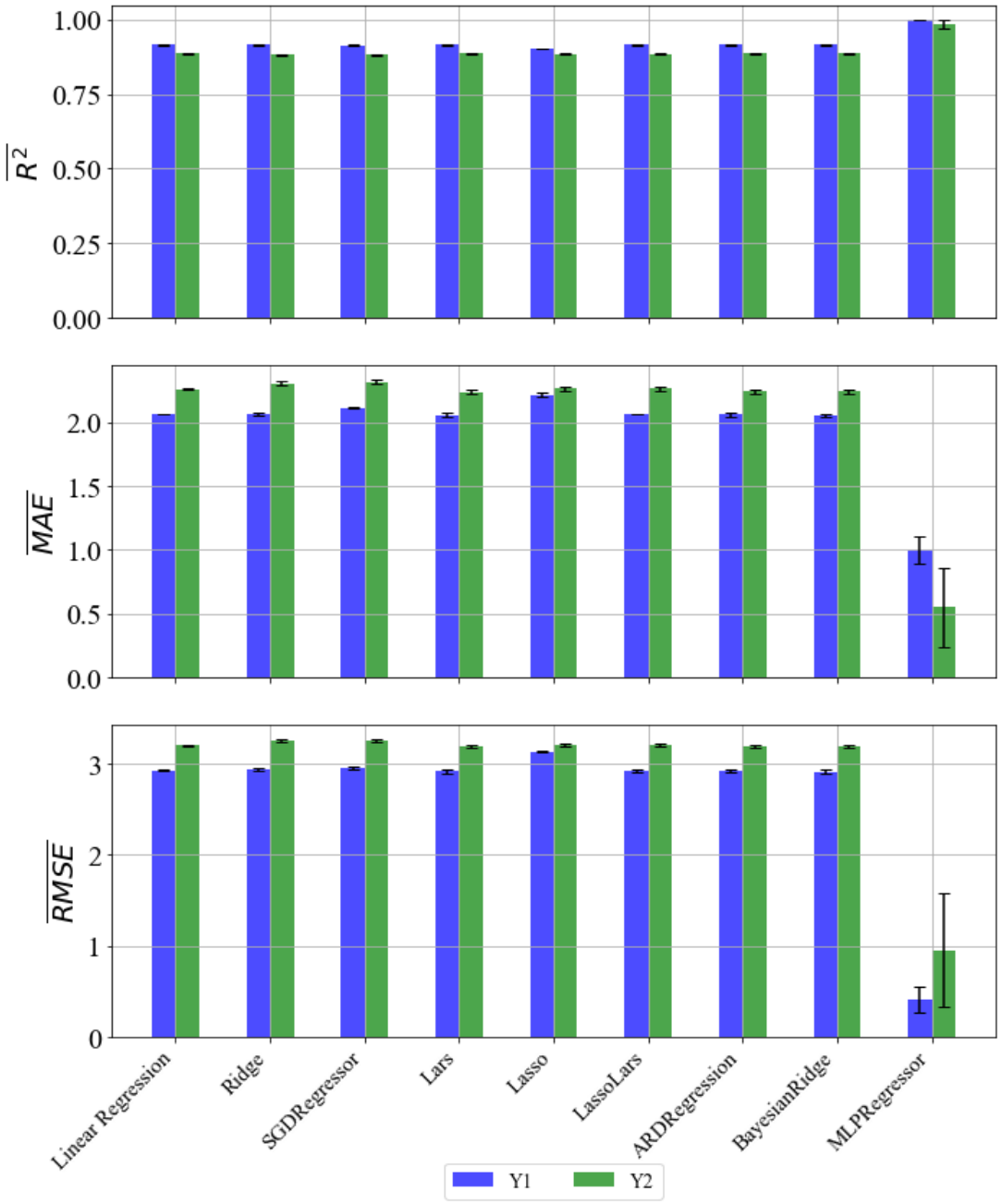
Tablica 8.5.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	300-400	396	300-500	302
tol	10^{-3}-10^{-2}	0.0013	10^{-5}-10^{-3}	0.0033
alpha_1	10^{-7}-10^{-5}	3.6371e-06	10^{-7}-10^{-5}	3.1007e-06
alpha_2	10^{-6}-10^{-5}	1.4846e-06	10^{-7}-10^{-5}	5.0819e-06
lambda_1	10^{-6}-10^{-5}	4.5751e-06	10^{-7}-10^{-5}	7.9086e-06
lambda_2	10^{-6}-10^{-5}	4.2095e-07	10^{-7}-10^{-5}	4.0476e-06
alpha_init	10^{-6}-10^{-5}	2.6651e-06	10^{-7}-10^{-5}	6.4707e-06
lambda_init	10^{-6}-10^{-5}	4.7458e-06	10^{-7}-10^{-5}	2.7743e-07
compute_score	/	true	/	true
fit_intercept	/	true	/	true
normalize	/	true	/	false
copy_X	/	true	/	false
verbose	/	true	/	false

Tablica 8.5.18. MLPRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	[86, 37, 20, 70, 75, 63, 59]	10-100	[39, 25, 10, 98]
activation	/	tanh	/	relu
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0004	10^{-5}-10^{-3}	0.0004
batch_size	do 200	30	do 200	102
learning_rate	/	adaptive	/	constant
learning_rate_init	10^{-5}-10^{-4}	1.4927e-05	10^{-5}-10^{-3}	6.0723e-05
power_t	0.3-0.7	0.5728	0.1-0.9	0.5898
max_iter	300-500	380	200-500	377
shuffle	/	true	/	false
random_state	0-150	94	0-100	23
tol	10^{-5}-10^{-3}	0.0003	10^{-5}-10^{-3}	0.0006
verbose	/	false	/	true
warm_start	/	false	/	true
momentum	0.2-0.8	0.5051	0.1-0.9	0.7243
nesterovs_momentum	/	true	/	true
early_stopping	/	true	/	false
validation_fraction	0.1-0.3	0.2779	0.1-0.3	0.2481
beta_1	0.7-0.9	0.7741	0.7-0.9	0.7814
beta_2	0.8-0.99	0.9411	0.8-0.99	0.8676
epsilon	10^{-9}-10^{-8}	1.1237e-09	10^{-9}-10^{-7}	8.002e-09
n_iter_no_change	5-20	8	5-20	13
max_fun	10000-20000	15734	10000-20000	17992

8.6. Rezultati dobiveni korištenjem StandardScaler-a



Slika 8.8. $\overline{R^2}$, $\overline{MAE}$, $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – StandardScaler

Usporedbom prethodnih rezultata sa onima iz poglavlja 7.6. vidimo da se vrijednosti nisu značajno promijenile nakon unakrsne validacije.

U nastavku će biti dane tablice koje sadrže raspone pretraživanja hiperparametara i vrijednosti onih hiperparametara koji su dali prethodne rezultate.

Tablica 8.6.10. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	False	/	False
n_jobs	30-70	47	1-100	94
positive	/	False	/	False

Tablica 8.6.11. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-10	2.9167	1-10	7.4878
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
max_iter	100-1000	199	100-1000	455
tol	10^{-4}-10^{-2}	0.0043	10^{-4}-10^{-2}	0.0076
solver	/	Saga	/	Saga
random_state	50-100	31	40-90	87

Tablica 8.6.12. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	squared_loss	/	squared_loss
alpha	0.001-0.03	0.0126	0.001-0.1	0.0058
fit_intercept	/	true	/	true
max_iter	100-1000	470	900-3000	251
penalty	/	elasticnet	/	elasticnet
l1_ratio	0.0-1.0	0.3896	0.2-1.0	0.1751
shuffle	/	True	/	true
verbose	0-0	0	0-0	0
epsilon	0.1-1.0	0.4919	0.01-0.1	0.8284
learning_rate	/	adaptive	/	invscaling
eta0	0.01-0.5	0.4235	0.01-0.1	0.1756
power_t	0.1-1.0	0.7579	0.1-1.0	0.6183
early_stopping	/	false	/	false
tol	10^{-3}-10^{-2}	0.0044	10^{-4}-10^{-3}	0.0012
random_state	1-50	6	0-100	1
Validation_fraction	0.1-1.0	0.3041	0.1-1.0	0.774
warm_start	/	False	/	True
average	/	false	/	false
n_iter_no_change	1-50	8	1-10	5

Tablica 8.6.13. Lars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	True
verbose	/	True	/	True
normalize	/	False	/	True
precompute	/	False	/	False
n_nonzero_coefs	10-100	46	20-120	64
eps	10^{-12}-10^{-10}	1.967e-11	10^{-4}-10^{-2}	6.454e-11
copy_X	/	True	/	True
fit_path	/	True	/	True
jitter	10^{-5}-10^{-4}	1.4776e-05	10^{-5}-10^{-3}	4.5546e-05
random_state	20-100	7	1-50	26

Tablica 8.6.14. Lasso – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-1.0	0.5349	0.01-0.1	0.0194
fit_intercept	/	True	/	True
normalize	/	False	/	False
precompute	/	True	/	False
copy_X	/	True	/	True
max_iter	3000-10000	335	4000-9000	926
tol	10^{-3}-10^{-2}	0.0002	10^{-3}-10^{-2}	0.0003
warm_start	/	True	/	False
positive	/	True	/	False
random_state	20-100	52	10-80	54

Tablica 8.6.15. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-1.0	0.004	0.1-1.0	0.0166
fit_intercept	/	True	/	True
verbose	/	False	/	False
normalize	/	False	/	False
precompute	/	True	/	False
max_iter	2000-6000	725	1500-5000	767
eps	/	0.0002	/	0.0005
copy_X	/	False	/	True
fit_path	/	False	/	False
positive	/	False	/	False
jitter	0.1-1.0	0.0103	0.01-0.1	0.0108
random_state	350-600	39	500-1000	46

Tablica 8.6.16. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	30-3000	339	50-1000	287
tol	10^{-4}-10^{-2}	0.0007	10^{-3}-10^{-2}	0.0008
alpha_1	10^{-6}-10^{-5}	0.0069	10^{-7}-10^{-5}	0.0013
alpha_2	10^{-6}-10^{-5}	7.7556e-07	10^{-6}-10^{-5}	8.7387e-06
lambda_1	10^{-7}-10^{-5}	3.1019e-07	10^{-6}-10^{-5}	9.4521e-07
lambda_2	10^{-7}-10^{-5}	7.1621e-06	10^{-6}-10^{-5}	1.5596e-06
compute_score	/	True	/	True
threshold_lambda	50000-100000	9134.8	50000-90000	10947.4
fit_intercept	/	True	/	True
normalize	/	False	/	True
copy_X	/	True	/	True
verbose	/	True	/	True

Tablica 8.6.17. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	1000-2500	334	200-1000	341
tol	10^{-3}-10^{-1}	0.0068	10^{-4}-10^{-2}	0.0016
alpha_1	10^{-5}-10^{-4}	2.0094e-06	10^{-6}-10^{-4}	7.077e-06
alpha_2	10^{-7}-10^{-4}	1.3045e-06	10^{-6}-10^{-4}	5.5908e-06
lambda_1	10^{-5}-10^{-4}	6.7003e-06	10^{-6}-10^{-4}	7.4305e-06
lambda_2	10^{-6}-10^{-4}	8.8631e-07	10^{-6}-10^{-4}	1.2449e-06
alpha_init	0.01-0.1	5.884e-06	0.1-1.0	4.3075e-06
lambda_init	0.001-0.1	8.8129e-06	0.001-0.1	8.3489e-06
compute_score	/	False	/	False
fit_intercept	/	True	/	True
normalize	/	False	/	False
copy_X	/	True	/	True
verbose	/	True	/	False

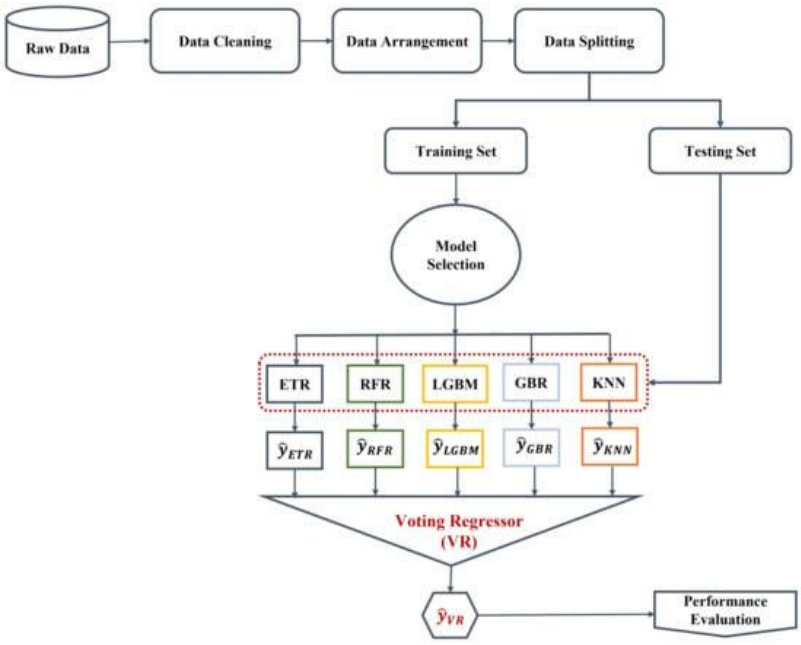
Tablica 8.6.18. MLPRegressor – Dobiveni hiperparametri

	Y1		**Y2**	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	[50, 40, 22, 90, 59, 69, 71, 51]	10-100	[87, 99, 12, 26, 86],
activation	/	relu	/	tanh
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0008	10^{-5}-10^{-3}	0.0004
batch_size	do 200	130	do 200	29
learning_rate	/	Constant	/	constant
learning_rate_init	10^{-5}-10^{-4}	3.2648e-05	10^{-5}-10^{-3}	7.5988e-05
power_t	0.3-0.7	0.5986	0.1-0.9	0.6211
max_iter	300-500	407	200-500	477
shuffle	/	False	/	false
random_state	0-150	12	0-100	66
tol	10^{-5}-10^{-3}	0.0001	10^{-5}-10^{-3}	0.0008
verbose	/	True	/	false
warm_start	/	True	/	false
momentum	0.2-0.8	0.4833	0.1-0.9	0.4505
nesterovs_momentum	/	True	/	true
early_stopping	/	False	/	true
validation_fraction	0.1-0.3	0.1745	0.1-0.3	0.1709
beta_1	0.7-0.9	0.88	0.7-0.9	0.7602
beta_2	0.8-0.99	0.8123	0.8-0.99	0.8219
epsilon	10^{-9}-10^{-8}	4.4106e-09	10^{-9}-10^{-7}	8.3726e-09
n_iter_no_change	5-20	11	5-20	5
max_fun	10000-20000	11951	10000-20000	12035

9. PRIMJENA ANSAMBL METODA

Ansambl u strojnom učenju je tehnika koja uključuje kombiniranje više modela kako bi se stvorio kvalitetniji i precizniji model za predviđanje. Budući da niti jedan model u strojnom učenju ne radi najbolje na svim problemima, tj. različiti modeli griješe na različite načine, ima ih smisla sjediniti. Osnovna ideja ansambl učenja je da kombiniranjem nekoliko modela prednosti svakog modela mogu kompenzirati nedostatke drugih što dovodi do bolje ukupne sposobnosti predikcije. Postoji nekolicina ansambl metoda neke od kojih su: bagging, boosting, stacking, glasanje, itd [36]. U ovom radu biti će primijenjene stacking metoda i glasanje.

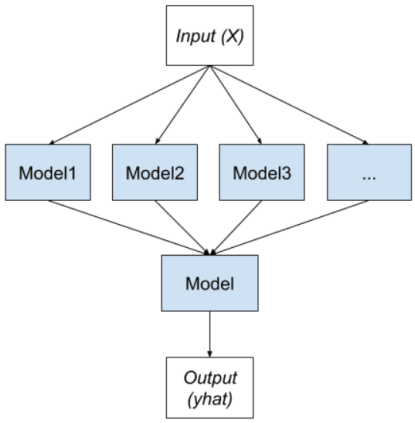
Glasanje (eng. voting) je ansambl metoda kod koje svi algoritmi daju svoja individualna predviđanja, a konačna vrijednost dobiva se ili putem većinskog glasanja (eng. hard voting) ili računanjem prosječnih vrijednosti predviđanja (eng. soft voting). Općenito se kao pristup odabire izračun prosječnih vrijednosti predviđanja. Na slici 9.1. vidimo primjer takve metode.



Slika 9.1. Primjer shematskog pregleda „VotingRegressor“ funkcije [37]

U ovom istraživanju korištena je funkcija *„VotingRegressor“* koja koristi više osnovnih modela za konačnu predikciju. Konačna predikcija jednaka je srednjoj vrijednosti predikcija svih osnovnih modela u ansamblu.

Sljedeća metoda koja će se koristiti je stacking metoda. Ova metoda ansambl učenja kombinira predviđanja više različitih modela kako bi se dobila konačna vrijednost. Koristi više slojeva modela u svrhu iskorištavanja prednosti svakog pojedinog modela i poboljšavanja ukupne performanse ansambla. Najčešće ima dva sloja, tj. razine, nulta razina i prva razina. Modeli razine nula su članovi ansambla, dok se modeli razine 1 obično koriste za kombiniranje predviđanja članova ansambla [38]. Općeniti shematski pregled modela ansambla sa dvije razine je prikazan na slici 9.2.



Slika 9.2. Općeniti shematski prikaz stacking ansambla [36]

U ovom istraživanju korištena je funkcija *„StackingRegressor“*. Ova funkcija daje opciju odabira finalnog estimatora koji će se koristiti za kombiniranje izlaza modela osnovne razine. U našem slučaju kao finalni estimator koristiti će se *„RidgeCV“*. Funkcija također pruža mogućnost provođenja unakrsne validacije, međutim ova opcija se neće koristiti.

Pri korištenju ansambl metodi od velike je važnosti ispravno odabrati modele koji će se koristiti u ansamblu. Važno je da modeli postižu čim bolje vrijednosti evaluacijskih metrika uz što manje vrijednosti standardne devijacije. U našem slučaju odabrati ćemo sve modele koji se dobiju nakon primjere skaliranja korištenjem *„StandardScaler-a“* jer svi ti modeli zadovoljavaju uvjete za korištenje u ansamblu.

Kod za ovaj proces napisan je na način da se definira funkcija koja provodi postupak izgradnje i evaluacije ansambla. Za svaki korišteni algoritam (npr. ARDRegression, MLPRegressor,…) prvo se nasumično biraju hiperparametri, izgrađuju se modeli za predikciju i dodaju u ansambl, a zatim se izvodi unakrsne validacija. Kada se postigne srednja vrijednost kvadratne greške veća od 0.99 ($\overline{R}^2 > 0.99$) model se ponovno trenira na podacima za trening i evaluira na podacima za testiranje.

U nastavku će biti dane tablice sa rezultatima postignutim primjenom stacking i voting ansambla.

Tablica 9.1. Voting ansambl

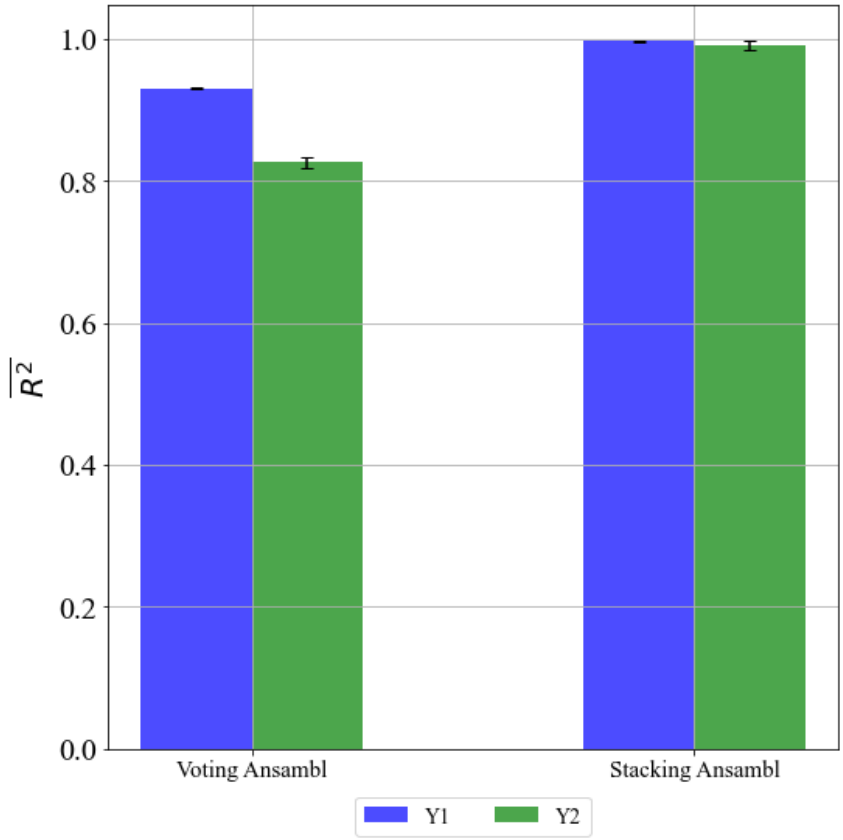
	Y1	**Y2**
$\overline{R^2}$ **(TRAIN)**	0.93254	0.83463
$\overline{R^2}$ **(TEST)**	0.92925	0.81842
$\overline{MAE}$ **(TRAIN)**	1.82924	3.02668
$\overline{MAE}$ **(TEST)**	1.86366	3.18778
$\overline{RMSE}$ **(TRAIN)**	2.61606	3.85911
$\overline{RMSE}$ **(TEST)**	2.65315	4.01148
$\mathbf{R^2}$ **(TEST)**	0.92785	0.80294
MAE (TEST)	1.92353	3.36562
RMSE (TEST)	2.70474	4.21454

Tablica 9.2. Stacking ansambl

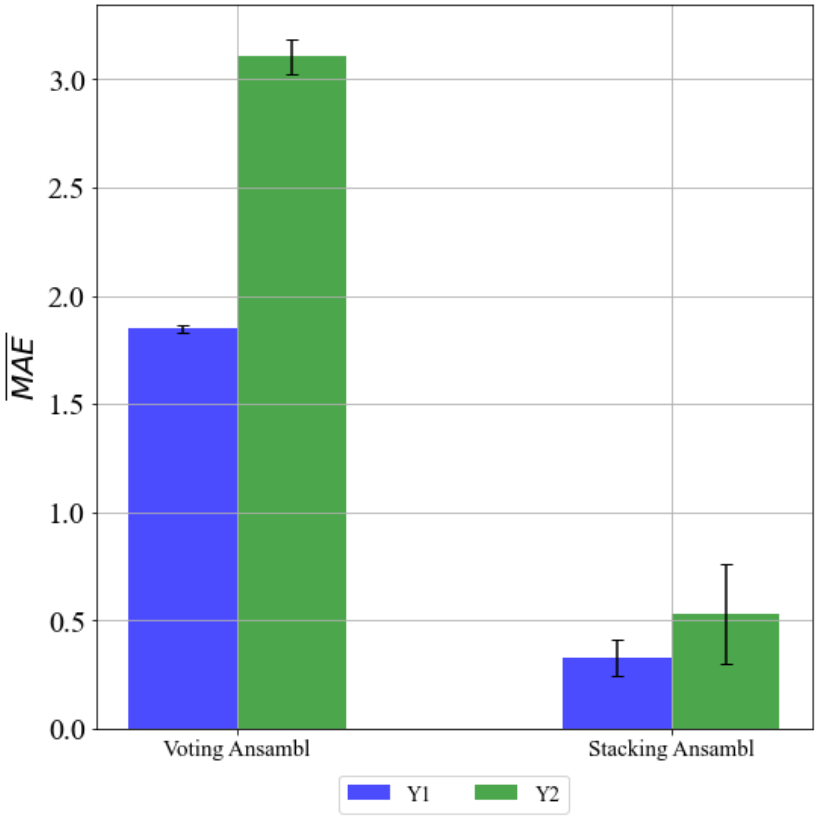
	Y1	**Y2**
$\overline{R^2}$ **(TRAIN)**	0.99882	0.99834
$\overline{R^2}$ **(TEST)**	0.99683	0.98539
$\overline{MAE}$ **(TRAIN)**	0.24564	0.29808
$\overline{MAE}$ **(TEST)**	0.41169	0.75814
$\overline{RMSE}$ **(TRAIN)**	0.34449	0.3848
$\overline{RMSE}$ **(TEST)**	0.56105	1.12764
$\mathbf{R^2}$ **(TEST)**	0.99727	0.99521
MAE (TEST)	0.37224	0.49519
RMSE (TEST)	0.52574	0.65723

Usporedbom rezultata iz tablica 9.1. i 9.2. vidimo da su primjenom stacking ansambla postignute puno bolje vrijednosti u usporedi sa primjenom voting ansambla. Razlog tome je što kako je prethodno opisano stacking ansambl kombinira različite modele kako bi se postigla najbolja moguća vrijednost, a voting ansambl kao konačan rezultat daje srednje vrijednosti svih korištenih modela. Ovime je završen proces kreiranja visokokvalitetnog prediktivnog modela za predikciju toplinskog dobitka i opterećenja prostora.

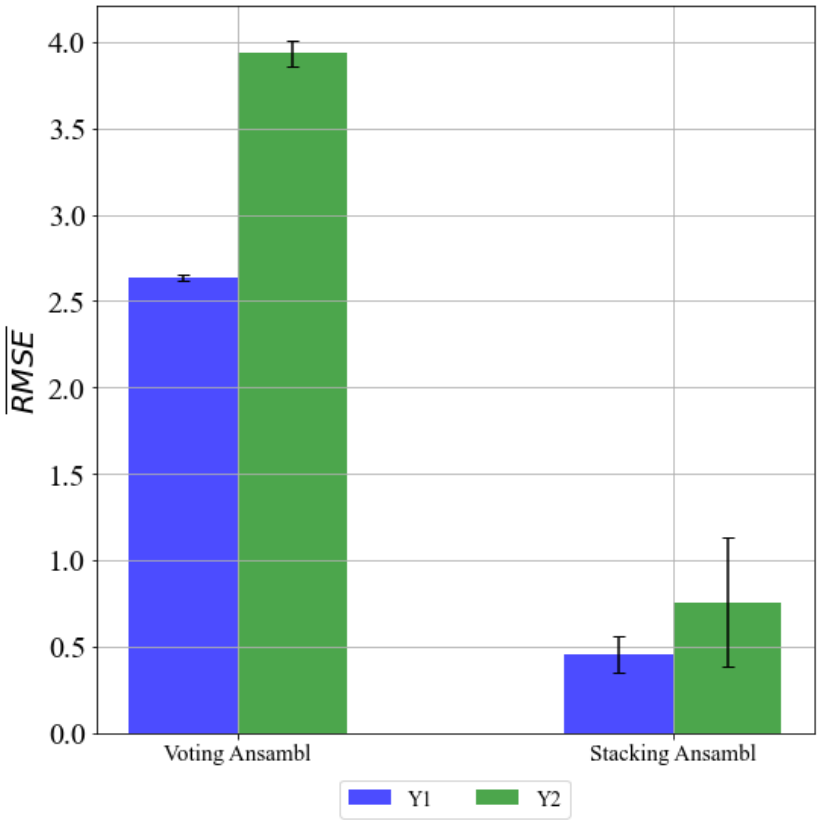
U nastavku će biti dane slike sa grafičkim prikazom rezultata evaluacijskih metrika za stacking i voting ansambl, te tablice sa hiperparametrima modela koji su korišteni u stacking ansambl.



Slika 9.3. $\overline{R^2}$ vrijednosti sa standardnom devijacijom – Voting i Stacking ansambl



Slika 9.4. $\overline{MAE}$ vrijednosti sa standardnom devijacijom – Voting i Stacking ansambl



Slika 9.5. $\overline{RMSE}$ vrijednosti sa standardnom devijacijom – Voting i Stacking ansambl

Tablica 9.3. LinearRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	False	/	False
normalize	/	True	/	False
copy_X	/	False	/	True
n_jobs	30-70	34	1-100	85
positive	/	False	/	True

Tablica 9.4. Ridge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	1-10	1.226	1-10	0.648
fit_intercept	/	False	/	True
normalize	/	True	/	False
copy_X	/	False	/	True
max_iter	100-1000	30	100-1000	95
tol	10^{-4}-10^{-2}	0.0007	10^{-4}-10^{-2}	0.0008
solver	/	auto	/	sag
random_state	50-100	4	40-90	6

Tablica 9.5. SGDRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
loss	/	epsilon_insensitive	/	epsilon_insensitive
alpha	0.001-0.03	0.0004	0.001-0.1	0.0008
fit_intercept	/	True	/	True
max_iter	100-1000	530	900-3000	603
penalty	/	elasticnet	/	l1
l1_ratio	0.0-1.0	0.155	0.2-1.0	0.2456
shuffle	/	True	/	True
verbose	0-0	0	0-0	0
epsilon	0.1-1.0	0.6907	0.01-0.1	0.3411
learning_rate	/	invscaling	/	constant
eta0	0.01-0.5	0.1985	0.01-0.1	0.2501
power_t	0.1-1.0	0.4761	0.1-1.0	0.625
early_stopping	/	False	/	False
tol	10^{-3}-10^{-2}	0.0014	10^{-4}-10^{-3}	0.0096
random_state	1-50	3	0-100	4
Validation_fraction	0.1-1.0	0.6602	0.1-1.0	0.9709
warm_start	/	True	/	True
average	/	True	/	True
n_iter_no_change	1-50	6	1-10	6

Tablica 9.6. Lars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
fit_intercept	/	True	/	False
verbose	/	False	/	False
normalize	/	False	/	False
precompute	/	False	/	auto
n_nonzero_coefs	10-100	58	20-120	19
eps	10^{-12}-10^{-10}	4.5032e-11	10^{-4}-10^{-2}	6.8265e-11
copy_X	/	True	/	True
fit_path	/	False	/	True
jitter	10^{-5}-10^{-4}	4.0086e-05	10^{-5}-10^{-3}	4.6202e-05
random_state	20-100	11	1-50	87

Tablica 9.7. Lasso – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.01-1.0	0.0982	0.01-0.1	0.8029
fit_intercept	/	True	/	False
normalize	/	False	/	True
precompute	/	True	/	False
copy_X	/	True	/	False
max_iter	3000-10000	465	4000-9000	922
tol	10^{-3}-10^{-2}	8.3396e-05	10^{-3}-10^{-2}	4.6862e-05
warm_start	/	True	/	True
positive	/	False	/	False
random_state	20-100	75	10-80	34

Tablica 9.8. LassoLars – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
alpha	0.1-1.0	0.0253	0.1-1.0	0.0102
fit_intercept	/	False	/	False
verbose	/	True	/	False
normalize	/	True	/	True
precompute	/	auto	/	auto
max_iter	2000-6000	529	1500-5000	628
eps	/	0.0009	/	0.0005
copy_X	/	True	/	True
fit_path	/	True	/	True
positive	/	False	/	False
jitter	0.1-1.0	0.083	0.01-0.1	0.0985
random_state	350-600	41	500-1000	87

Tablica 9.9. ARDRegression – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	30-3000	279	50-1000	328
tol	10^{-4}-10^{-2}	0.0001	10^{-3}-10^{-2}	0.0006
alpha_1	10^{-6}-10^{-5}	0.0076	10^{-7}-10^{-5}	0.0013
alpha_2	10^{-6}-10^{-5}	2.0051e-06	10^{-6}-10^{-5}	2.4527e-06
lambda_1	10^{-7}-10^{-5}	6.4214e-07	10^{-6}-10^{-5}	3.4818e-07
lambda_2	10^{-7}-10^{-5}	6.2622e-06	10^{-6}-10^{-5}	3.9196e-06
compute_score	/	True	/	True
threshold_lambda	50000-100000	10855.1	50000-90000	10062.5
fit_intercept	/	False	/	False
normalize	/	False	/	True
copy_X	/	True	/	True
verbose	/	False	/	True

Tablica 9.10. BayesianRidge – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
n_iter	1000-2500	392	200-1000	393
tol	10^{-3}-10^{-1}	0.0064	10^{-4}-10^{-2}	0.0001
alpha_1	10^{-5}-10^{-4}	1.9376e-06	10^{-6}-10^{-4}	5.3254e-06
alpha_2	10^{-7}-10^{-4}	4.2694e-06	10^{-6}-10^{-4}	5.5065e-06
lambda_1	10^{-5}-10^{-4}	3.3651e-06	10^{-6}-10^{-4}	1.9232e-06
lambda_2	10^{-6}-10^{-4}	1.5688e-06	10^{-6}-10^{-4}	2.2838e-06
alpha_init	0.01-0.1	2.3083e-06	0.1-1.0	5.1208e-06
lambda_init	0.001-0.1	1.4379e-06	0.001-0.1	4.2699e-06
compute_score	/	False	/	True
fit_intercept	/	False	/	False
normalize	/	False	/	False
copy_X	/	False	/	True
verbose	/	False	/	True

Tablica 9.11. MLPRegressor – Dobiveni hiperparametri

	Y1		Y2	
Ime	**Raspon**	**Vrijednost**	**Raspon**	**Vrijednost**
hidden_layer_sizes	10-100	(18, 42, 33, 98, 77, 41, 73, 36, 72)	10-100	(58, 99, 24, 34)
activation	/	relu	/	relu
solver	/	lbfgs	/	lbfgs
alpha	10^{-5}-10^{-3}	0.0004	10^{-5}-10^{-3}	0.0002
batch_size	do 200	189	do 200	22
learning_rate	/	constant	/	adaptive
learning_rate_init	10^{-5}-10^{-4}	8.5131e-05	10^{-5}-10^{-3}	9.8214e-05
power_t	0.3-0.7	0.6364	0.1-0.9	0.3108
max_iter	300-500	400	200-500	409
shuffle	/	True	/	False
random_state	0-150	28	0-100	18
tol	10^{-5}-10^{-3}	0.0003	10^{-5}-10^{-3}	0.0003
verbose	/	True	/	False
warm_start	/	True	/	True
momentum	0.2-0.8	0.5728	0.1-0.9	0.5295
nesterovs_momentum	/	False	/	False
early_stopping	/	True	/	False
validation_fraction	0.1-0.3	0.2029	0.1-0.3	0.1265
beta_1	0.7-0.9	0.7042	0.7-0.9	0.8014
beta_2	0.8-0.99	0.8671	0.8-0.99	0.8194
epsilon	10^{-9}-10^{-8}	8.8574e-09	10^{-9}-10^{-7}	3.7917e-09
n_iter_no_change	5-20	10	5-20	17
max_fun	10000-20000	17968	10000-20000	13321

10. ZAKLJUČAK

Na javno dostupnom skupu podataka koji sadrži osam različitih karakteristika zgrada, te dva rezultirajuća odziva (toplinski dobitak i toplinsko opterećenje prostora) provedene su statistička i korelacijska analiza. Nakon toga su koristeći devet različitih algoritama razvijeni modeli za predviđanje oba odziva. Sljedeći korak bio je ispitati utjecaj različitih metoda skaliranja i normalizacije na rezultate predikcije korištenih algoritama. Idući korak je bio dodatno poboljšanje performansi modela kroz postupak nasumičnog pretraživanja hiperparametara na skupu podataka sa i bez primjene metodi skaliranja i normalizacije. Nasumično pretraživanje hiperparametara sa unakrsnom validacijom korišteno je da bi se izbjegla pretreniranost modela. Posljednji korak u ovom procesu je bio izbor optimalnih algoritama za implementaciju u ansambl tehnike poput stacking-a i voting-a.

Ovaj sveobuhvatan pristup omogućio je stvaranje visokokvalitetnih prediktivnih modela koji kombiniraju snagu različitih algoritama, te osiguravaju pouzdane rezultate u različitim scenarijima. Rezultat ovog rada podržava izvedivost korištenja alata strojnog učenja za procjenu parametara zgrada kao praktičnog i preciznog pristupa.

Prednosti korištenog pristupa uključuju stvaranje visokokvalitetnih modela za predikciju koji kombiniraju različite algoritme i osiguravaju pouzdane rezultate. Ovi modeli mogu biti od koristi projektantima i inženjerima pri odabiru opreme za grijanje i hlađenje u energetski učinkovitim zgradama, što može doprinijeti smanjenju energetske potrošnje. Također postoje i neki nedostaci, kao što je potreba za velikim skupom podataka za treniranje modela, te za dodatnim vremenskim i financijskim resursima za implementaciju ovakvog pristupa.

Primjena metodi skaliranja i normalizacije dovela je do značajnog poboljšanja prediktivnih sposobnosti modela za određene algoritme u usporedbi sa inicijalnom analizom, a za one algoritme kod kojih se predikcija nije poboljšala nije primijećeno pogoršanje prediktivnih sposobnosti. Također, vrijeme izračuna se nije promijenilo nakon implementacije skaliranja i normalizacije. Nasumičnim pretraživanjem hiperparametara dodatno su se poboljšale prediktivne sposobnosti određenih modela, bez da su se pogoršale sposobnosti ostalih. No ovaj postupak se pokazao vremenski zahtjevnim i to posebno u slučajevima korištenja algoritama „MLPRegressor“, „SGDRegressor“ i „BayesianRidge“. Razlog tomu je veliki broj hiperparametara koje je potrebno pretraživati, te čije je raspone pretraživanja potrebno ručno

podešavati. Posebno loše pokazalo se korištenje „SGDRegressor“ i „BayesianRidge“ algoritama, jer nisu dali bolje rezultate unatoč velikim vremenskim i računalnim zahtjevima za njihovu upotrebu. Nasumično pretraživanje hiperparametara sa unakrsnom validacijom iziskivalo je jednako puno vremena kao i pretraživanje hiperparametara bez unakrsne validacije, međutim ovaj pristup je ukazao na problem pretreniranosti određenih modela. Računalno najintenzivniji proces je bio primjena voting ansambla. Zbog načina rada te metode potrebno je prvo odrediti individualne predikcije svih korištenih modela čiji se prosjek zatim računa. Taj postupak se pokazao vremenski neisplativim jer je tražio više vremena od primjene stacking ansambla, a dao znatno lošije rezultate. Stacking ansambl je opravdao svoju isplativost jer je unatoč velikim vremenskim zahtjevima dao izuzetno dobre rezultate.

Istraživanje bi se moglo proširiti razmatrajući dodatne karakteristike zgrada koje bi mogle utjecati na energetske performanse, kao i primjenjujući drugih tehnika strojnog učenja i ansambl metoda koje bi mogle dalje poboljšati modele za predikciju. Također je važno istražiti primjenu ovih modela na stvarnim projektima izgradnje energetski učinkovitih zgrada kako bi se potvrdila njihova praktična primjenjivost i preciznost u stvarnim scenarijima.

11. LITERATURA

1. Liu J, Chang H, Forrest JY, Yang B. Influence of artificial intelligence on technological innovation: Evidence from the panel data of china's manufacturing sectors. Technological Forecasting and Social Change. 2020 Sep 1;158:120142.
2. Sun H, Edziah BK, Kporsu AK, Sarkodie SA, Taghizadeh-Hesary F. Energy efficiency: The role of technological innovation and knowledge spillover. Technological Forecasting and Social Change. 2021 Jun 1;167:120659.
3. Capgemini Research Institute. Climate AI: How Artificial Intelligence Can Power Your Climate Action Strategy. Capgemini US. 2020.
4. S interneta, https://www.statista.com/statistics/1022326/worldwide-ai-impact-ghg/
5. Wang J, Qv D, Ni L, Fan J, Kong W. Matching-design for inverter air-source heat pump system based on heating load characteristics of civil buildings. Energy and Buildings. 2022 Apr 1;260:111952.
6. Dong B, Cao C, Lee SE. Applying support vector machines to predict building energy consumption in tropical region. Energy and Buildings. 2005 May 1;37(5):545-53.
7. Zhang J, Haghighat F. Development of Artificial Neural Network based heat convection algorithm for thermal simulation of large rectangular cross-sectional area Earth-to-Air Heat Exchangers. Energy and Buildings. 2010 Apr 1;42(4):435-40.
8. Tsanas A, Xifara A. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy and buildings. 2012 Jun 1;49:560-7.
9. Catalina T, Virgone J, Blanco E. Development and validation of regression models to predict monthly heating demand for residential buildings. Energy and buildings. 2008 Jan 1;40(10):1825-32.
10. Sedgwick P. Pearson's correlation coefficient. Bmj. 2012 Jul 4;345.
11. S interneta, https://www.ibm.com/topics/machine-learning
12. Massaron L, Boschetti A. Regression analysis with Python. Packt Publishing Ltd; 2016 Feb 29.
13. Rokem A, Kay K. Fractional ridge regression: a fast, interpretable reparameterization of ridge regression. GigaScience. 2020 Dec;9(12):giaa133.

14. Shi, A. SGDRegressor with Scikit-Learn: Untaught Lessons You Need to Know. 2023. Available online: https://towardsdatascience.com/sgdregressor-with-scikit-learn-untaught-lessons-you-need-to-know-cf2430439689#:~:text=The%20SGDRegressor%20algorithm%20uses%20stochastic,with%20respect%20to%20the%20parameters
15. Efron B, Hastie T, Johnstone I, Tibshirani R. Least angle regression.
16. S interneta, https://www.datacamp.com/tutorial/tutorial-lasso-ridge-regression
17. S interneta, https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LassoLars.html#sklearn.linear_model.LassoLars
18. S interneta, https://www.datatechnotes.com/2020/10/regression-example-with-ardregression-in-python.html
19. S interneta, https://www.tutorialspoint.com/scikit_learn/scikit_learn_bayesian_ridge_regression.htm
20. S interneta, https://vitalflux.com/sklearn-neural-network-regression-example-mlpregressor/
21. Hahn GJ. The coefficient of determination exposed. Chemtech. 1973 Oct;3(10):609-12.
22. Willmott CJ, Matsuura K. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate research. 2005 Dec 19;30(1):79-82.
23. Anđelić N, Lorencin I, Glučina M, Car Z. Mean Phase Voltages and Duty Cycles Estimation of a Three-Phase Inverter in a Drive System Using Machine Learning Algorithms. Electronics. 2022 Aug 21;11(16):2623.
24. Dietterich T. Overfitting and undercomputing in machine learning. ACM computing surveys (CSUR). 1995 Sep 1;27(3):326-7.
25. Singh D, Singh B. Investigating the impact of data normalization on classification performance. Applied Soft Computing. 2020 Dec 1;97:105524.
26. Abd Halim KN, Jaya AS, Fadzil AF. Data pre-processing algorithm for neural network binary classification model in bank tele-Marketing. International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2020;9:272-7.
27. Singh P, Singh P. MLlib: Machine Learning Library. Learn PySpark: Build Python-based Machine Learning and Deep Learning Models. 2019:85-115.

28. S interneta, https://wellsr.com/python/data-scaling-and-normalization-with-python/#:~:text=In%20standard%20scaling%2C%20a%20feature,standard%20deviation%20of%20the%20data.&text=Here%2C%20u%20refers%20to%20the,corresponds%20to%20the%20standard%20deviation.
29. Probst P, Boulesteix AL, Bischl B. Tunability: Importance of hyperparameters of machine learning algorithms. The Journal of Machine Learning Research. 2019 Jan 1;20(1):1934-65.
30. S interneta, https://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html
31. S interneta, https://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
32. Bergstra J, Bengio Y. Random search for hyper-parameter optimization. Journal of machine learning research. 2012 Feb 1;13(2).
33. Berrar D. Cross-Validation.
34. S interneta, https://rforhr.com/kfold.html
35. Manna S. K-Fold Cross validationfor deep learning models using keras.
36. Brownlee J. A gentle introduction to ensemble learning algorithms. Machine Learning Mastery. 2021 Apr;27.
37. Phyo PP, Byun YC, Park N. Short-term energy forecasting using machine-learning-based ensemble voting regression. Symmetry. 2022 Jan 14;14(1):160.
38. Divina F, Gilson A, Goméz-Vela F, García Torres M, Torres JF. Stacking ensemble learning for short-term electricity consumption forecasting. Energies. 2018 Apr 16;11(4):949.

12. SAŽETAK

Realiziran je visokokvalitetni model za predikciju toplinskog dobitka i toplinskog opterećenja prostora korištenjem tehnika strojnog učenja. Na skupu podataka napravljene su statistička i korelacijska analiza. Napravljena je inicijalna analiza modela sa zadanim hiperparametrima, nakon čega su primijenjene metode skaliranja i normalizacije. Zatim je napravljen kod za nasumično pretraživanje hiperparametara na skupu podataka sa i bez primjene metodi skaliranja i normalizacije. Proces nasumičnog pretraživanja hiperparametara kombiniran je sa unakrsnom validacijom. Za kraj su primijenjene određene ansambl tehnike kako bi se dobio robustan model za predikciju.

Ključne riječi: Statistička analiza, Korelacijska analiza, Skaliranje, Normalizacija, Nasumično pretraživanje hiperparametara, Unakrsna validacija, Ansambl.

13. ABSTRACT

A high-quality model for predicting the heating and cooling loads of a space has been developed using machine learning techniques. Statistical and correlation analyses were conducted on the dataset. An initial analysis of the model was performed with default hyperparameters, followed by the application of scaling and normalization methods. Afterwards, code was developed for random hyperparameter search on both scaled and normalized datasets and the dataset without scaling and normalization. The process of random hyperparameter search was combined with cross-validation. Finally, specific ensemble techniques were applied to obtain a robust predictive model.

Keywords: Statistical analysis, Correlation analysis, Scaling, Normalization, Hyperparameter tuning, Cross-validation, Ensemble.

DODATAK A

```
# Statisticka i korelacijska analiza

# Umetanje potrebnih knjiznica
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt

pd.set_option("mode.sim_interactive", True)
pd.get_option("display.max_columns")
pd.set_option("display.max_columns", 10)

# Umetanje dataseta
# Koristen je dataset sa stranice: https://www.kaggle.com/datasets/elikplim/eergy-efficiency-dataset

path = r"C:\Users\tomev\Desktop\Izborni projekt\dataset.csv"
data = pd.read_csv(path) #Dataset za gadanje Y1

path_b = r"C:\Users\tomev\Desktop\Izborni projekt\dataset.csv"
data_b = pd.read_csv(path_b) #Dataset za gadanje Y2

# Statisticka analiza

print('Statisticka analiza', data.describe())

# Korelacijska analiza

print('Korelacijska analiza', data.corr())

plt.figure(figsize = (10,10))
dataplot = sb.heatmap(data.corr(), cmap="YlGnBu", annot=True)
plt.show()
```

DODATAK B

```
# Inicijalna analiza za Y1

# Umetanje potrebnih knjiznica
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

#Umetanje algoritama
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor

# Umetanje evaluacijskih metoda
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

pd.set_option("mode.sim_interactive", True)
pd.get_option("display.max_columns")
pd.set_option("display.max_columns", 10)

# 1. Umetanje dataseta
# Koristen je dataset sa stranice: https://www.kaggle.com/datasets/elikplim/eergy-efficiency-dataset

path = r"C:\Users\tomev\Desktop\Faks\2. Semestar\Izborni projekt\dataset.csv"
data = pd.read_csv(path) #Dataset za gadanje Y1

path_b = r"C:\Users\tomev\Desktop\Faks\2. Semestar\Izborni projekt\dataset.csv"
data_b = pd.read_csv(path_b) #Dataset za gadanje Y2

# 2.A Gadamo Y1
print('Gadamo Y1')
print('')
data2 = data.pop('Y2') # Izvucen Y2
y = data.pop('Y1') # Izvucen Y1
X = data # Definiran X

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)

#Linear regression
print('Linearna regresija (Y1)')
lin_regr = LinearRegression()
reg = LinearRegression().fit(X, y)
#R^2
R2Train_Lin_regr = r2_score(y_train, reg.predict(X_train))
print("R2Train_Lin_regr = {}".format(R2Train_Lin_regr))
R2Test_Lin_regr = r2_score(y_test, reg.predict(X_test))
print("R2Test_Lin_regr = {}".format(R2Test_Lin_regr))
```

```
#Mean Absolute error
MAETrain_Lin_regr = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_Lin_regr = {}".format(MAETrain_Lin_regr))
MAETest_Lin_regr = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_Lin_regr = {}".format(MAETest_Lin_regr))
#Root mean squared error
RMSTrain_Lin_regr = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_Lin_regr = {}".format(RMSTrain_Lin_regr))
RMSTest_Lin_regr = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_Lin_regr = {}".format(RMSTest_Lin_regr))
print('')

print('Ridge (Y1)')
#Ridge
clf = Ridge(alpha=1.0)
clf.fit(X, y)
#R^2
R2Train_Ridge = r2_score(y_train, clf.predict(X_train))
print("R2Train_Ridge = {}".format(R2Train_Ridge))
R2Test_Ridge = r2_score(y_test, clf.predict(X_test))
print("R2Test_Ridge = {}".format(R2Test_Ridge))
#Mean Absolute error
MAETrain_Ridge = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_Ridge = {}".format(MAETrain_Ridge))
MAETest_Ridge = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_Ridge = {}".format(MAETest_Ridge))
#Root mean squared error
RMSTrain_Ridge = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_Ridge = {}".format(RMSTrain_Ridge))
RMSTest_Ridge = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_Ridge = {}".format(RMSTest_Ridge))
print('')

print('SGDRegressor (Y1)')
#SGDRegressor
reg = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
reg.fit(X, y)
#R^2
R2Train_SDGR = r2_score(y_train, reg.predict(X_train))
print("R2Train_SDGR = {}".format(R2Train_SDGR))
R2Test_SDGR = r2_score(y_test, reg.predict(X_test))
print("R2Test_SDGR = {}".format(R2Test_SDGR))
#Mean Absolute error
MAETrain_SDGR = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_SDGR = {}".format(MAETrain_SDGR))
MAETest_SDGR = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_SDGR = {}".format(MAETest_SDGR))
#Root mean squared error
RMSTrain_SDGR = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_SDGR = {}".format(RMSTrain_SDGR))
RMSTest_SDGR = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_SDGR = {}".format(RMSTest_SDGR))
print('')

print('Lars (Y1)')
```

```
#Lars
reg = linear_model.Lars(n_nonzero_coefs=1, normalize=False)
reg.fit(X,y)
#R^2
R2Train_Lars = r2_score(y_train, reg.predict(X_train))
print("R2Train_Lars = {}".format(R2Train_Lars))
R2Test_Lars = r2_score(y_test, reg.predict(X_test))
print("R2Test_Lars = {}".format(R2Test_Lars))
#Mean Absolute error
MAETrain_Lars = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_Lars = {}".format(MAETrain_Lars))
MAETest_Lars = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_Lars = {}".format(MAETest_Lars))
#Root mean squared error
RMSTrain_Lars = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_Lars = {}".format(RMSTrain_Lars))
RMSTest_Lars = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_Lars = {}".format(RMSTest_Lars))
print('')

print('Lasso (Y1)')
#Lasso
clf = linear_model.Lasso(alpha=0.1)
clf.fit(X,y)
#R^2
R2Train_Lasso = r2_score(y_train, clf.predict(X_train))
print("R2Train_Lasso = {}".format(R2Train_Lasso))
R2Test_Lasso = r2_score(y_test, clf.predict(X_test))
print("R2Test_Lasso = {}".format(R2Test_Lasso))
#Mean Absolute error
MAETrain_Lasso = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_Lasso = {}".format(MAETrain_Lasso))
MAETest_Lasso = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_Lasso = {}".format(MAETest_Lasso))
#Root mean squared error
RMSTrain_Lasso = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_Lasso = {}".format(RMSTrain_Lasso))
RMSTest_Lasso = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_Lasso = {}".format(RMSTest_Lasso))
print('')


print('LassoLars (Y1)')
#LassoLars
reg = linear_model.LassoLars(alpha=0.01, normalize=False)
reg.fit(X,y)
#R^2
R2Train_LassoLars = r2_score(y_train, reg.predict(X_train))
print("R2Train_LassoLars = {}".format(R2Train_LassoLars))
R2Test_LassoLars = r2_score(y_test, reg.predict(X_test))
print("R2Test_LassoLars = {}".format(R2Test_LassoLars))
#Mean Absolute error
MAETrain_LassoLars = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_LassoLars = {}".format(MAETrain_LassoLars))
MAETest_LassoLars = mean_absolute_error(y_test, reg.predict(X_test))
```

```
print("MAETest_LassoLars = {}".format(MAETest_LassoLars))
#Root mean squared error
RMSTrain_LassoLars = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_LassoLars = {}".format(RMSTrain_LassoLars))
RMSTest_LassoLars = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_LassoLars = {}".format(RMSTest_LassoLars))
print('')

print('ARDRegression (Y1)')
#ARDRegresion
clf = linear_model.ARDRegression()
clf.fit(X,y)
#R^2
R2Train_ARDR = r2_score(y_train, clf.predict(X_train))
print("R2Train_ARDR = {}".format(R2Train_ARDR))
R2Test_ARDR = r2_score(y_test, clf.predict(X_test))
print("R2Test_ARDR = {}".format(R2Test_ARDR))
#Mean Absolute error
MAETrain_ARDR = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_ARDR = {}".format(MAETrain_ARDR))
MAETest_ARDR = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_ARDR = {}".format(MAETest_ARDR))
#Root mean squared error
RMSTrain_ARDR = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_ARDR = {}".format(RMSTrain_ARDR))
RMSTest_ARDR = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_ARDR = {}".format(RMSTest_ARDR))
print('')

print('BayesianRidge (Y1)')
#BayesianRidge
clf = linear_model.BayesianRidge()
clf.fit(X,y)
#R^2
R2Train_BayRidge = r2_score(y_train, clf.predict(X_train))
print("R2Train_BayRidge = {}".format(R2Train_BayRidge))
R2Test_BayRidge = r2_score(y_test, clf.predict(X_test))
print("R2Test_BayRidge = {}".format(R2Test_BayRidge))
#Mean Absolute error
MAETrain_BayRidge = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_BayRidge = {}".format(MAETrain_BayRidge))
MAETest_BayRidge = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_BayRidge = {}".format(MAETest_BayRidge))
#Root mean squared error
RMSTrain_BayRidge = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_BayRidge = {}".format(RMSTrain_BayRidge))
RMSTest_BayRidge = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_BayRidge = {}".format(RMSTest_BayRidge))
print('')

print('MLPRegressor (Y1)')
#MLPRegressor
regr = MLPRegressor(max_iter=2000)
regr.fit(X_train,y_train)
#R^2
```

```
R2Train_MLPR = r2_score(y_train, regr.predict(X_train))
print("R2Train_MLPR = {}".format(R2Train_MLPR))
R2Test_MLPR = r2_score(y_test, regr.predict(X_test))
print("R2Test_MLPR = {}".format(R2Test_MLPR))
#Mean Absolute error
MAETrain_MLPR = mean_absolute_error(y_train, regr.predict(X_train))
print("MAETrain_MLPR = {}".format(MAETrain_MLPR))
MAETest_MLPR = mean_absolute_error(y_test, regr.predict(X_test))
print("MAETest_MLPR = {}".format(MAETest_MLPR))
#Root mean squared error
RMSTrain_MLPR = np.sqrt(mean_squared_error(y_train, regr.predict(X_train)))
print("RMSTrain_MLPR = {}".format(RMSTrain_MLPR))
RMSTest_MLPR = np.sqrt(mean_squared_error(y_test, regr.predict(X_test)))
print("RMSTest_MLPR = {}".format(RMSTest_MLPR))
print('')
```

DODATAK C

```
# Analiza i rezultati dobiveni koristenjem MaxAbsScaler-a za Y1

# Umetanje potrebnih knjiznica
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

#Umetanje algoritama
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn.neural_network import MLPRegressor

# Umetanje evaluacijskih metoda
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

# Umetanje metode za skaliranje
from sklearn.preprocessing import MaxAbsScaler

pd.set_option("mode.sim_interactive", True)
pd.get_option("display.max_columns")
pd.set_option("display.max_columns", 10)

# 1. Umetanje dataseta
# Koristen je dataset sa stranice: https://www.kaggle.com/datasets/elikplim/eergy-efficiency-dataset

path = r"C:\Users\tomev\Desktop\dataset.csv"
data = pd.read_csv(path) #Dataset za gadanje Y1

path_b = r"C:\Users\tomev\Desktop\dataset.csv"
data_b = pd.read_csv(path_b) #Dataset za gadanje Y2

# 2.A Gadamo Y1
print('Gadamo Y1')
print('')
data2 = data.pop('Y2') # Izvucen Y2
y = data.pop('Y1') # Izvucen Y1
X = data # Definiran X

scaler = MaxAbsScaler()
X = scaler.fit_transform(X) # Skaliranje ulaznih varijabli

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)

#Linear regression
print('Linearna regresija (Y1)')
lin_regr = LinearRegression()
```

```
reg = LinearRegression().fit(X, y)
#R^2
R2Train_Lin_regr = r2_score(y_train, reg.predict(X_train))
print("R2Train_Lin_regr = {}".format(R2Train_Lin_regr))
R2Test_Lin_regr = r2_score(y_test, reg.predict(X_test))
print("R2Test_Lin_regr = {}".format(R2Test_Lin_regr))
#Mean Absolute error
MAETrain_Lin_regr = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_Lin_regr = {}".format(MAETrain_Lin_regr))
MAETest_Lin_regr = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_Lin_regr = {}".format(MAETest_Lin_regr))
#Root mean squared error
RMSTrain_Lin_regr = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_Lin_regr = {}".format(RMSTrain_Lin_regr))
RMSTest_Lin_regr = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_Lin_regr = {}".format(RMSTest_Lin_regr))
print('')

print('Ridge (Y1)')
#Ridge
clf = Ridge(alpha=1.0)
clf.fit(X, y)
#R^2
R2Train_Ridge = r2_score(y_train, clf.predict(X_train))
print("R2Train_Ridge = {}".format(R2Train_Ridge))
R2Test_Ridge = r2_score(y_test, clf.predict(X_test))
print("R2Test_Ridge = {}".format(R2Test_Ridge))
#Mean Absolute error
MAETrain_Ridge = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_Ridge = {}".format(MAETrain_Ridge))
MAETest_Ridge = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_Ridge = {}".format(MAETest_Ridge))
#Root mean squared error
RMSTrain_Ridge = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_Ridge = {}".format(RMSTrain_Ridge))
RMSTest_Ridge = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_Ridge = {}".format(RMSTest_Ridge))
print('')

print('SGDRegressor (Y1)')
#SGDRegressor
reg = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
reg.fit(X, y)
#R^2
R2Train_SDGR = r2_score(y_train, reg.predict(X_train))
print("R2Train_SDGR = {}".format(R2Train_SDGR))
R2Test_SDGR = r2_score(y_test, reg.predict(X_test))
print("R2Test_SDGR = {}".format(R2Test_SDGR))
#Mean Absolute error
MAETrain_SDGR = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_SDGR = {}".format(MAETrain_SDGR))
MAETest_SDGR = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_SDGR = {}".format(MAETest_SDGR))
#Root mean squared error
RMSTrain_SDGR = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
```

```
print("RMSTrain_SDGR = {}".format(RMSTrain_SDGR))
RMSTest_SDGR = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_SDGR = {}".format(RMSTest_SDGR))
print('')

print('Lars (Y1)')
#Lars
reg = linear_model.Lars(n_nonzero_coefs=1, normalize=False)
reg.fit(X,y)
#R^2
R2Train_Lars = r2_score(y_train, reg.predict(X_train))
print("R2Train_Lars = {}".format(R2Train_Lars))
R2Test_Lars = r2_score(y_test, reg.predict(X_test))
print("R2Test_Lars = {}".format(R2Test_Lars))
#Mean Absolute error
MAETrain_Lars = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_Lars = {}".format(MAETrain_Lars))
MAETest_Lars = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_Lars = {}".format(MAETest_Lars))
#Root mean squared error
RMSTrain_Lars = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_Lars = {}".format(RMSTrain_Lars))
RMSTest_Lars = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_Lars = {}".format(RMSTest_Lars))
print('')

print('Lasso (Y1)')
#Lasso
clf = linear_model.Lasso(alpha=0.1)
clf.fit(X,y)
#R^2
R2Train_Lasso = r2_score(y_train, clf.predict(X_train))
print("R2Train_Lasso = {}".format(R2Train_Lasso))
R2Test_Lasso = r2_score(y_test, clf.predict(X_test))
print("R2Test_Lasso = {}".format(R2Test_Lasso))
#Mean Absolute error
MAETrain_Lasso = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_Lasso = {}".format(MAETrain_Lasso))
MAETest_Lasso = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_Lasso = {}".format(MAETest_Lasso))
#Root mean squared error
RMSTrain_Lasso = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_Lasso = {}".format(RMSTrain_Lasso))
RMSTest_Lasso = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_Lasso = {}".format(RMSTest_Lasso))
print('')


print('LassoLars (Y1)')
#LassoLars
reg = linear_model.LassoLars(alpha=0.01, normalize=False)
reg.fit(X,y)
#R^2
R2Train_LassoLars = r2_score(y_train, reg.predict(X_train))
print("R2Train_LassoLars = {}".format(R2Train_LassoLars))
```

```
R2Test_LassoLars = r2_score(y_test, reg.predict(X_test))
print("R2Test_LassoLars = {}".format(R2Test_LassoLars))
#Mean Absolute error
MAETrain_LassoLars = mean_absolute_error(y_train, reg.predict(X_train))
print("MAETrain_LassoLars = {}".format(MAETrain_LassoLars))
MAETest_LassoLars = mean_absolute_error(y_test, reg.predict(X_test))
print("MAETest_LassoLars = {}".format(MAETest_LassoLars))
#Root mean squared error
RMSTrain_LassoLars = np.sqrt(mean_squared_error(y_train, reg.predict(X_train)))
print("RMSTrain_LassoLars = {}".format(RMSTrain_LassoLars))
RMSTest_LassoLars = np.sqrt(mean_squared_error(y_test, reg.predict(X_test)))
print("RMSTest_LassoLars = {}".format(RMSTest_LassoLars))
print('')

print('ARDRegression (Y1)')
#ARDRegresion
clf = linear_model.ARDRegression()
clf.fit(X,y)
#R^2
R2Train_ARDR = r2_score(y_train, clf.predict(X_train))
print("R2Train_ARDR = {}".format(R2Train_ARDR))
R2Test_ARDR = r2_score(y_test, clf.predict(X_test))
print("R2Test_ARDR = {}".format(R2Test_ARDR))
#Mean Absolute error
MAETrain_ARDR = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_ARDR = {}".format(MAETrain_ARDR))
MAETest_ARDR = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_ARDR = {}".format(MAETest_ARDR))
#Root mean squared error
RMSTrain_ARDR = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_ARDR = {}".format(RMSTrain_ARDR))
RMSTest_ARDR = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_ARDR = {}".format(RMSTest_ARDR))
print('')

print('BayesianRidge (Y1)')
#BayesianRidge
clf = linear_model.BayesianRidge()
clf.fit(X,y)
#R^2
R2Train_BayRidge = r2_score(y_train, clf.predict(X_train))
print("R2Train_BayRidge = {}".format(R2Train_BayRidge))
R2Test_BayRidge = r2_score(y_test, clf.predict(X_test))
print("R2Test_BayRidge = {}".format(R2Test_BayRidge))
#Mean Absolute error
MAETrain_BayRidge = mean_absolute_error(y_train, clf.predict(X_train))
print("MAETrain_BayRidge = {}".format(MAETrain_BayRidge))
MAETest_BayRidge = mean_absolute_error(y_test, clf.predict(X_test))
print("MAETest_BayRidge = {}".format(MAETest_BayRidge))
#Root mean squared error
RMSTrain_BayRidge = np.sqrt(mean_squared_error(y_train, clf.predict(X_train)))
print("RMSTrain_BayRidge = {}".format(RMSTrain_BayRidge))
RMSTest_BayRidge = np.sqrt(mean_squared_error(y_test, clf.predict(X_test)))
print("RMSTest_BayRidge = {}".format(RMSTest_BayRidge))
print('')
```

```
print('MLPRegressor (Y1)')
#MLPRegressor
regr = MLPRegressor(max_iter=2000)
regr.fit(X_train,y_train)
#R^2
R2Train_MLPR = r2_score(y_train, regr.predict(X_train))
print("R2Train_MLPR = {}".format(R2Train_MLPR))
R2Test_MLPR = r2_score(y_test, regr.predict(X_test))
print("R2Test_MLPR = {}".format(R2Test_MLPR))
#Mean Absolute error
MAETrain_MLPR = mean_absolute_error(y_train, regr.predict(X_train))
print("MAETrain_MLPR = {}".format(MAETrain_MLPR))
MAETest_MLPR = mean_absolute_error(y_test, regr.predict(X_test))
print("MAETest_MLPR = {}".format(MAETest_MLPR))
#Root mean squared error
RMSTrain_MLPR = np.sqrt(mean_squared_error(y_train, regr.predict(X_train)))
print("RMSTrain_MLPR = {}".format(RMSTrain_MLPR))
RMSTest_MLPR = np.sqrt(mean_squared_error(y_test, regr.predict(X_test)))
print("RMSTest_MLPR = {}".format(RMSTest_MLPR))

mean_R2_LinearRegressionY1 = np.mean([R2Train_Lin_regr, R2Test_Lin_regr])
mean_MAE_LinearRegressionY1 = np.mean([MAETrain_Lin_regr, MAETest_Lin_regr])
mean_RMSE_LinearRegressionY1 = np.mean([RMSTrain_Lin_regr, RMSTest_Lin_regr])

std_R2_LinearRegressionY1 = np.std([R2Train_Lin_regr, R2Test_Lin_regr])
std_MAE_LinearRegressionY1 = np.std([MAETrain_Lin_regr, MAETest_Lin_regr])
std_RMSE_LinearRegressionY1 = np.std([RMSTrain_Lin_regr, RMSTest_Lin_regr])

mean_R2_RidgeY1 = np.mean([R2Train_Ridge, R2Test_Ridge])
mean_MAE_RidgeY1 = np.mean([MAETrain_Ridge, MAETest_Ridge])
mean_RMSE_RidgeY1 = np.mean([RMSTrain_Ridge, RMSTest_Ridge])

std_R2_RidgeY1 = np.std([R2Train_Ridge, R2Test_Ridge])
std_MAE_RidgeY1 = np.std([MAETrain_Ridge, MAETest_Ridge])
std_RMSE_RidgeY1 = np.std([RMSTrain_Ridge, RMSTest_Ridge])

mean_R2_SGDRY1 = np.mean([R2Train_SDGR, R2Test_SDGR])
mean_MAE_SGDRY1 = np.mean([MAETrain_SDGR, MAETest_SDGR])
mean_RMSE_SGDRY1 = np.mean([RMSTrain_SDGR, RMSTest_SDGR])

std_R2_SGDRY1 = np.std([R2Train_SDGR, R2Test_SDGR])
std_MAE_SGDRY1 = np.std([MAETrain_SDGR, MAETest_SDGR])
std_RMSE_SGDRY1 = np.std([RMSTrain_SDGR, RMSTest_SDGR])

mean_R2_LarsY1 = np.mean([R2Train_Lars, R2Test_Lars])
mean_MAE_LarsY1 = np.mean([MAETrain_Lars, MAETest_Lars])
mean_RMSE_LarsY1 = np.mean([RMSTrain_Lars, RMSTest_Lars])

std_R2_LarsY1 = np.std([R2Train_Lars, R2Test_Lars])
std_MAE_LarsY1 = np.std([MAETrain_Lars, MAETest_Lars])
std_RMSELarsY1 = np.std([RMSTrain_Lars, RMSTest_Lars])

mean_R2_LassoY1 = np.mean([R2Train_Lasso, R2Test_Lasso])
mean_MAE_LassoY1 = np.mean([MAETrain_Lasso, MAETest_Lasso])
```

```
mean_RMSE_LassoY1 = np.mean([RMSTrain_Lasso, RMSTest_Lasso])

std_R2_LassoY1 = np.std([R2Train_Lasso, R2Test_Lasso])
std_MAE_LassoY1 = np.std([MAETrain_Lasso, MAETest_Lasso])
std_RMSE_LassoY1 = np.std([RMSTrain_Lasso, RMSTest_Lasso])

mean_R2_LassoLarsY1 = np.mean([R2Train_LassoLars, R2Test_LassoLars])
mean_MAE_LassoLarsY1 = np.mean([MAETrain_LassoLars, MAETest_LassoLars])
mean_RMSE_LassoLarsY1 = np.mean([RMSTrain_LassoLars, RMSTest_LassoLars])

std_R2_LassoLarsY1 = np.std([R2Train_LassoLars, R2Test_LassoLars])
std_MAE_LassoLarsY1 = np.std([MAETrain_LassoLars, MAETest_LassoLars])
std_RMSE_LassoLarsY1 = np.std([RMSTrain_LassoLars, RMSTest_LassoLars])

mean_R2_ARDRY1 = np.mean([R2Train_ARDR, R2Test_ARDR])
mean_MAE_ARDRY1 = np.mean([MAETrain_ARDR, MAETest_ARDR])
mean_RMSE_ARDRY1 = np.mean([RMSTrain_ARDR, RMSTest_ARDR])

std_R2_ARDRY1 = np.std([R2Train_ARDR, R2Test_ARDR])
std_MAE_ARDRY1 = np.std([MAETrain_ARDR, MAETest_ARDR])
std_RMSE_ARDRY1 = np.std([RMSTrain_ARDR, RMSTest_ARDR])

mean_R2_BayRidgeY1 = np.mean([R2Train_BayRidge, R2Test_BayRidge])
mean_MAE_BayRidgeY1 = np.mean([MAETrain_BayRidge, MAETest_BayRidge])
mean_RMSE_BayRidgeY1 = np.mean([RMSTrain_BayRidge, RMSTest_BayRidge])

std_R2_BayRidgeY1 = np.std([R2Train_BayRidge, R2Test_BayRidge])
std_MAE_BayRidgeY1 = np.std([MAETrain_BayRidge, MAETest_BayRidge])
std_RMSE_BayRidgeY1 = np.std([RMSTrain_BayRidge, RMSTest_BayRidge])

mean_R2_MLPRY1 = np.mean([R2Train_MLPR, R2Test_MLPR])
mean_MAE_MLPRY1 = np.mean([MAETrain_MLPR, MAETest_MLPR])
mean_RMSE_MLPRY1 = np.mean([RMSTrain_MLPR, RMSTest_MLPR])

std_R2_MLPRY1 = np.std([R2Train_MLPR, R2Test_MLPR])
std_MAE_MLPRY1 = np.std([MAETrain_MLPR, MAETest_MLPR])
std_RMSE_MLPRY1 = np.std([RMSTrain_MLPR, RMSTest_MLPR])
```

DODATAK D

```
# Nasumicno pretrazivanje hiperparametara
# LinearRegression, Y1

# Umetanje potrebnih knjiznica
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import random
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# LinearRegression
path = r"C:\Users\tomev\Desktop\dataset.csv"
data_a = pd.read_csv(path) # Dataset za gadanje Y1
data_b = pd.read_csv(path) # Dataset za gadanje Y1

# Gadamo Y1
print('Gadamo Y1')
data2_a = data_a.pop('Y2') # Izvucen Y2
y = data_a.pop('Y1') # Izvucen Y1
X = data_a # Definiran X

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

file_path_LinearRegression_a1 =
r"C:\Users\tomev\Desktop\LinearRegression_Rezultati\LinearRegressionY1_Hiperparametri.txt"
file_path_LinearRegression_a2 =
r"C:\Users\tomev\Desktop\LinearRegression_Rezultati\LinearRegressionY1_DobiveneVrijednosti.txt"

def random_search():
    parameters = {}
    parameters['fit_intercept'] = random.choice([True, False])
    parameters['normalize'] = random.choice([True, False])
    parameters['copy_X'] = random.choice([True, False])
    parameters['n_jobs'] = random.randint(1,100)
    parameters['positive'] = random.choice([True, False])
    print("Parameters:", parameters)
    return parameters

def linear_regression(parameter, X_train, y_train, X_test, y_test):
    lin_reg = LinearRegression(fit_intercept=parameter['fit_intercept'],
                        normalize=parameter['normalize'],
                        copy_X=parameter['copy_X'],
                        n_jobs=parameter['n_jobs'],
                        positive=parameter['positive'])
    lin_reg.fit(X_train, y_train)

    R2Train = r2_score(y_train, lin_reg.predict(X_train))
    R2Test = r2_score(y_test, lin_reg.predict(X_test))
    MAETrain = mean_absolute_error(y_train, lin_reg.predict(X_train))
    MAETest = mean_absolute_error(y_test, lin_reg.predict(X_test))
    RMSETrain = np.sqrt(mean_squared_error(y_train, lin_reg.predict(X_train)))
```

```
    RMSETest = np.sqrt(mean_squared_error(y_test, lin_reg.predict(X_test)))
    MeanR2 = np.mean([R2Train, R2Test])
    MeanMAE = np.mean([MAETrain, MAETest])
    MeanRMSE = np.mean([RMSETrain, RMSETest])
    STDR2 = np.std([R2Train, R2Test])
    STDMAE = np.std([MAETrain, MAETest])
    STDRMSE = np.std([RMSETrain, RMSETest])

    print("R2Train = {}".format(R2Train))
    print("R2Test = {}".format(R2Test))
    print("MAETrain = {}".format(MAETrain))
    print("MAETest = {}".format(MAETest))
    print("RMSETrain = {}".format(RMSETrain))
    print("RMSETest = {}".format(RMSETest))
    print("MeanR2 = {}".format(MeanR2))
    print("MeanMAE = {}".format(MeanMAE))
    print("MeanRMSE = {}".format(MeanRMSE))
    print("STDR2 = {}".format(STDR2))
    print("STDMAE = {}".format(STDMAE))
    print("STDRMSE = {}".format(STDRMSE))

    with open(file_path_LinearRegression_a2, "w") as file:
        file.write("Dobivene vrijednosti za Y1:\n")
        file.write("R2Train = {}\n".format(R2Train))
        file.write("R2Test = {}\n".format(R2Test))
        file.write("MAETrain = {}\n".format(MAETrain))
        file.write("MAETest = {}\n".format(MAETest))
        file.write("RMSETrain = {}\n".format(RMSETrain))
        file.write("RMSETest = {}\n".format(RMSETest))
        file.write("MeanR2 = {}\n".format(MeanR2))
        file.write("MeanMAE = {}\n".format(MeanMAE))
        file.write("MeanRMSE = {}\n".format(MeanRMSE))
        file.write("STDR2 = {}\n".format(STDR2))
        file.write("STDMAE = {}\n".format(STDMAE))
        file.write("STDRMSE = {}\n".format(STDRMSE))

    return MeanR2
k = 0
while True:
    print("Broj iteracije = {}".format(k))
    try:
        param = random_search()
        res = linear_regression(param, X_train, y_train, X_test, y_test)
        k += 1
        if res > 0.915:
            print("Pronađeno rješenje!")
            print("Rezultat = {}".format(res))
            with open(file_path_LinearRegression_a1, "w") as file:
                file.write("Odabrani hiperparametri za Y1:\n")
                file.write("fit_intercept = {}\n".format(param['fit_intercept']))
                file.write("normalize = {}\n".format(param['normalize']))
                file.write("copy_X = {}\n".format(param['copy_X']))
                file.write("n_jobs = {}\n".format(param['n_jobs']))
                file.write("positive = {}\n".format(param['positive']))
            break
```

```
        print("Rezultat = {}".format(res))
    except ZeroDivisionError:
        print("Dijeljenje s nulom!")
```

DODATAK E

```
# Nasumicno pretrazivanje hiperparametara i krosvalidacija
# Bez skaliranja i normalizacije
# LinearRegression, Y1

# Umetanje potrebnih knjiznica
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import random
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_validate
import json

# LinearRegression
path = r"C:\Users\tomev\Desktop\dataset.csv"
data_a = pd.read_csv(path) # Dataset za gadanje Y1
data_b = pd.read_csv(path) # Dataset za gadanje Y1

# Gadamo Y1
print('Gadamo Y1')
data2_a = data_a.pop('Y2') # Izvucen Y2
y = data_a.pop('Y1') # Izvucen Y1
X = data_a # Definiran X

file_path_vrijednosti =
r"C:\Users\tomev\Desktop\LinearRegression_Rezultati\LinearRegression_BezSkaliranjaY1_Vrijednosti.txt"
file_path_parametri =
r"C:\Users\tomev\Desktop\LinearRegression_Rezultati\LinearRegression_BezSkaliranjaY1_Parametri.txt"

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

def random_search():
    parameters = {
        'fit_intercept': random.choice([True, False]),
        'normalize': random.choice([True, False]),
        'copy_X': random.choice([True, False]),
        'n_jobs': random.randint(1, 100),
        'positive': random.choice([True, False])
    }
    print("Parameters:", parameters)
    return parameters

def LinearReg(parameters, X_train,y_train,X_test, y_test):
    model = LinearRegression(**parameters)
    cvmodel = cross_validate(model, X_train, y_train, cv=5,
                             scoring = ("r2",
                                        "neg_mean_absolute_error",
                                        "neg_root_mean_squared_error",
                                        "neg_mean_absolute_percentage_error"),
                             return_train_score=True)
```

```
    print("All Scores From CV5 = {}".format(cvmodel))
    with open(file_path_vrijednosti, "w") as file1:
        file1.write("R2 Train Scores = {}\n".format(cvmodel['train_r2']))
        file1.write("R2 Test Scores = {}\n".format(cvmodel['test_r2']))
        file1.write("MAE Train Scores = {}\n".format(abs(cvmodel['train_neg_mean_absolute_error'])))
        file1.write("MAE Test Scores = {}\n".format(abs(cvmodel['test_neg_mean_absolute_error'])))
        file1.write("RMSE Train Scores = {}\n".format(abs(cvmodel['train_neg_root_mean_squared_error'])))
        file1.write("RMSE Test Scores = {}\n".format(abs(cvmodel['test_neg_root_mean_squared_error'])))
        file1.write("MAPE Train Scores =
{}\n".format(abs(cvmodel['train_neg_mean_absolute_percentage_error'])))
        file1.write("MAPE Test Scores =
{}\n".format(abs(cvmodel['test_neg_mean_absolute_percentage_error'])))

    AvrR2ScoreTrain = np.mean(cvmodel['train_r2'])
    StdR2ScoreTrain = np.std(cvmodel['train_r2'])
    AvrR2ScoreTest = np.mean(cvmodel['test_r2'])
    StdR2ScoreTest = np.std(cvmodel['test_r2'])
    AvrAllR2Score = np.mean([AvrR2ScoreTrain,AvrR2ScoreTest])
    StdAllR2Score = np.std([AvrR2ScoreTrain, AvrR2ScoreTest])

    AvrMAEScoreTrain = np.mean(abs(cvmodel['train_neg_mean_absolute_error']))
    StdMAEScoreTrain = np.std(abs(cvmodel['train_neg_mean_absolute_error']))
    AvrMAEScoreTest = np.mean(abs(cvmodel['test_neg_mean_absolute_error']))
    StdMAEScoreTest = np.std(abs(cvmodel['test_neg_mean_absolute_error']))
    AvrAllMAEScore = np.mean([AvrMAEScoreTrain, AvrMAEScoreTest])
    StdAllMAEScore = np.std([AvrMAEScoreTrain, AvrMAEScoreTest])

    AvrRMSEScoreTrain = np.mean(abs(cvmodel['train_neg_root_mean_squared_error']))
    StdRMSEScoreTrain = np.std(abs(cvmodel['train_neg_root_mean_squared_error']))
    AvrRMSEScoreTest = np.mean(abs(cvmodel['test_neg_root_mean_squared_error']))
    StdRMSEScoreTest = np.std(abs(cvmodel['test_neg_root_mean_squared_error']))
    AvrAllRMSEScore = np.mean([AvrRMSEScoreTrain, AvrRMSEScoreTest])
    StdAllRMSEScore = np.std([AvrRMSEScoreTrain, AvrRMSEScoreTest])

    with open(file_path_vrijednosti, "a") as file1:
        file1.write("############################################################\n"+\
                "AvrR2Score Train = {}\n".format(AvrR2ScoreTrain)+\
                "StdR2Score Train = {}\n".format(StdR2ScoreTrain)+\
                "AvrR2Score Test = {}\n".format(AvrR2ScoreTest)+\
                "StdR2Score Test = {}\n".format(StdR2ScoreTest)+\
                "AvrMAEScore Train = {}\n".format(AvrMAEScoreTrain)+\
                "StdMAEScore Train = {}\n".format(StdMAEScoreTrain)+\
                "AvrMAEScore Test = {}\n".format(AvrMAEScoreTest)+\
                "StdMAEScore Test ={}\n".format(StdMAEScoreTest)+\
                "AvrRMSEScore Train = {}\n".format(AvrRMSEScoreTrain)+\
                "StdRMSEScore Train = {}\n".format(StdRMSEScoreTrain)+\
                "AvrRMSEScore Test = {}\n".format(AvrRMSEScoreTest)+\
                "StdRMSEScore Test = {}\n".format(StdRMSEScoreTest)+\
                "############################################################\n"
                "mean_R2_LinearRegressionY1 = {}\n".format(AvrAllR2Score)+\
                "std_R2_LinearRegressionY1 = {}\n".format(StdAllR2Score)+\
                "mean_MAE_LinearRegressionY1 = {}\n".format(AvrAllMAEScore)+\
                "std_MAE_LinearRegressionY1 = {}\n".format(StdAllMAEScore)+\
                "mean_RMSE_LinearRegressionY1 = {}\n".format(AvrAllRMSEScore)+\
                "std_RMSE_LinearRegressionY1 = {}\n".format(StdAllRMSEScore)+\
```

```
                "#################################################################\n")

    if AvrAllR2Score > 0.91:
        with open(file_path_vrijednosti, "a") as file1:
            file1.write("#################################################################\n")
            file1.write(" Final Evaluation\n")
            file1.write("#################################################################\n")
        model.fit(X_train,y_train)
        R2Test = model.score(X_test,y_test)
        MAETest = mean_absolute_error(y_test, model.predict(X_test))
        RMSETest = np.sqrt(mean_squared_error(y_test, model.predict(X_test)))
        print("R^2 Test = {}".format(R2Test))
        print("MAE Test = {}".format(MAETest))
        print("RMSE Test = {}".format(RMSETest))
        with open(file_path_vrijednosti, "a") as file1:
            file1.write("#########################################################\n")
            file1.write("model R^2 Test = {}\n".format(R2Test))
            file1.write("MAE Test = {}\n".format(MAETest))
            file1.write("RMSE Test = {}\n".format(RMSETest))
            file1.write("#########################################################\n")
        return R2Test
    else:
        return AvrAllR2Score

k = 0
while True:
    print("Current Iteration = {}".format(k))
    LinParam = random_search()
    test = LinearReg(LinParam,X_train,y_train, X_test, y_test)
    k+=1
    if test > 0.91:
        with open(file_path_parametri, "w") as file2:
            file2.write("Parametri\n")
            json.dump(LinParam, file2)
        print("Solution is Found!!")
        break
    else:
        continue
```

DODATAK F

```
# Stacking ansambl, Y1

# Umetanje potrebnih knjiznica
from sklearn.ensemble import StackingRegressor
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import random
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_validate

from sklearn.linear_model import ARDRegression
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import Lars
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoLars
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import SGDRegressor

path = r"C:\Users\tomev\Desktop\dataset.csv"
data_a = pd.read_csv(path) # Dataset za gadanje Y1
data_b = pd.read_csv(path) # Dataset za gadanje Y1

# Gadamo Y1
print('Gadamo Y1')
data2_a = data_a.pop('Y2') # Izvucen Y2
y = data_a.pop('Y1') # Izvucen Y1
X = data_a # Definiran X

scaler = StandardScaler()
X = scaler.fit_transform(X) # Skaliranje ulaznih varijabli

file_path_param_stackingY2 = r"C:\Users\tomev\Desktop\Stacking_Rezultati\Stacking_HiperparametriY1.txt"

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

def StackingEnsemble (X_train, X_test, y_train, y_test):

    # Nasumicni odabir parametara

    def random_search_ardr():
        parameters_ardr = {}
        parameters_ardr['n_iter'] = random.randint(250, 350)
        parameters_ardr['tol'] = random.uniform(1e-5, 1e-3)
        parameters_ardr['alpha_1'] = random.uniform(1e-2, 1e-5)
        parameters_ardr['alpha_2'] = random.uniform(1e-8, 1e-5)
        parameters_ardr['lambda_1'] = random.uniform(1e-7, 1e-6)
        parameters_ardr['lambda_2'] = random.uniform(1e-6, 1e-5)
        parameters_ardr['compute_score'] = random.choice([True, False])
```

```
        parameters_ardr['threshold_lambda'] = random.uniform(9000, 11000)
        parameters_ardr['fit_intercept'] = random.choice([True, False])
        parameters_ardr['normalize'] = random.choice([True, False])
        parameters_ardr['copy_X'] = random.choice([True, False])
        parameters_ardr['verbose'] = random.choice([True, False])
        print("Parameters:", parameters_ardr)
        with open(file_path_param_stackingY2, "w") as f:
            f.write("parameters_ardr:\n")
            f.write(f"{parameters_ardr}: {parameters_ardr}\n")
        return parameters_ardr

    def random_search_bayridge():
        parameters_bayridge = {}
        parameters_bayridge['n_iter'] = random.randint(300, 400)
        parameters_bayridge['tol'] = random.uniform(1e-4, 1e-2)
        parameters_bayridge['alpha_1'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['alpha_2'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['lambda_1'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['lambda_2'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['alpha_init'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['lambda_init'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['compute_score'] = random.choice([True, False])
        parameters_bayridge['fit_intercept'] = random.choice([True, False])
        parameters_bayridge['normalize'] = random.choice([True, False])
        parameters_bayridge['copy_X'] = random.choice([True, False])
        parameters_bayridge['verbose'] = random.choice([True, False])
        print("Parameters:", parameters_bayridge)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_bayridge:\n")
            f.write(f"{parameters_bayridge}: {parameters_bayridge}\n")
        return parameters_bayridge

    def random_search_lars():
        parameters_lars = {}
        parameters_lars['fit_intercept'] = random.choice([True, False])
        parameters_lars['verbose'] = random.choice([True, False])
        parameters_lars['normalize'] = random.choice([True, False])
        parameters_lars['precompute'] = random.choice(['auto', True, False])
        parameters_lars['n_nonzero_coefs'] = random.randint(10, 100)
        parameters_lars['eps'] = random.uniform(1e-16, 1e-10)
        parameters_lars['copy_X'] = random.choice([True, False])
        parameters_lars['fit_path'] = random.choice([True, False])
        parameters_lars['jitter'] = random.uniform(1e-10, 1e-4)
        parameters_lars['random_state'] = random.randint(1, 100)
        print("Parameters:", parameters_lars)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_lars:\n")
            f.write(f"{parameters_lars}: {parameters_lars}\n")
        return parameters_lars

    def random_search_lasso():
        parameters_lasso = {}
        parameters_lasso['alpha'] = random.uniform(0.01, 1.0)
        parameters_lasso['fit_intercept'] = random.choice([True, False])
        parameters_lasso['normalize'] = random.choice([True, False])
```

```
        parameters_lasso['precompute'] = random.choice([True, False])
        parameters_lasso['copy_X'] = random.choice([True, False])
        parameters_lasso['max_iter'] = random.randint(100, 1000)
        parameters_lasso['tol'] = random.uniform(1e-6, 1e-3)
        parameters_lasso['warm_start'] = random.choice([True, False])
        parameters_lasso['positive'] = random.choice([True, False])
        parameters_lasso['random_state'] = random.randint(1, 100)
        parameters_lasso['selection'] = random.choice(['cyclic', 'random'])
        print("Parameters:", parameters_lasso)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_lasso:\n")
            f.write(f"{parameters_lasso}: {parameters_lasso}\n")
        return parameters_lasso

    def random_search_lassolars():
        parameters_lassolars = {}
        parameters_lassolars['alpha'] = random.uniform(0.001, 0.1)
        parameters_lassolars['fit_intercept'] = random.choice([True, False])
        parameters_lassolars['verbose'] = random.choice([True, False])
        parameters_lassolars['normalize'] = random.choice([True, False])
        parameters_lassolars['precompute'] = random.choice([True, False, 'auto'])
        parameters_lassolars['max_iter'] = random.randint(100, 1000)
        parameters_lassolars['eps'] = random.uniform(np.finfo(float).eps, 1e-3)
        parameters_lassolars['copy_X'] = random.choice([True, False])
        parameters_lassolars['fit_path'] = random.choice([True, False])
        parameters_lassolars['positive'] = random.choice([True, False])
        parameters_lassolars['jitter'] = random.uniform(0.01, 0.1)
        parameters_lassolars['random_state'] = random.randint(1, 100)
        print("Parameters:", parameters_lassolars)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_lassolars:\n")
            f.write(f"{parameters_lassolars}: {parameters_lassolars}\n")
        return parameters_lassolars

    def random_search_linreg():
        parameters_linreg = {
        'fit_intercept': random.choice([True, False]),
        'normalize': random.choice([True, False]),
        'copy_X': random.choice([True, False]),
        'n_jobs': random.randint(1, 100),
        'positive': random.choice([True, False])
        }
        print("Parameters:", parameters_linreg)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_linreg:\n")
            f.write(f"{parameters_linreg}: {parameters_linreg}\n")
        return parameters_linreg

    def random_search_mlp():
        parameters_mlp = {}
        numHidLayers = random.randint(2, 10)
        hidLayers = []
        for i in range(1, numHidLayers, 1):
            hidLayers.append(random.randint(10, 100))
        tupleHid = tuple(hidLayers)
```

```
        parameters_mlp['hidden_layer_sizes'] = tupleHid
        parameters_mlp['activation'] = random.choice(['identity', 'logistic', 'tanh', 'relu'])
        parameters_mlp['solver'] = random.choice(['lbfgs', 'sgd', 'adam'])
        parameters_mlp['alpha'] = random.uniform(1e-5, 1e-3)
        parameters_mlp['batch_size'] = random.randint(1, min(X_train.shape[0], 200))
        parameters_mlp['learning_rate'] = random.choice(['constant', 'invscaling', 'adaptive'])
        parameters_mlp['learning_rate_init'] = random.uniform(1e-5, 1e-4)
        parameters_mlp['power_t'] = random.uniform(0.3, 0.7)
        parameters_mlp['max_iter'] = random.randint(300, 500)
        parameters_mlp['shuffle'] = random.choice([True, False])
        parameters_mlp['random_state'] = random.randint(0, 150)
        parameters_mlp['tol'] = random.uniform(1e-5, 1e-3)
        parameters_mlp['verbose'] = random.choice([True, False])
        parameters_mlp['warm_start'] = random.choice([True, False])
        parameters_mlp['momentum'] = random.uniform(0.2, 0.8)
        parameters_mlp['nesterovs_momentum'] = random.choice([True, False])
        parameters_mlp['early_stopping'] = random.choice([True, False])
        parameters_mlp['validation_fraction'] = random.uniform(0.1, 0.3)
        parameters_mlp['beta_1'] = random.uniform(0.7, 0.9)
        parameters_mlp['beta_2'] = random.uniform(0.8, 0.99)
        parameters_mlp['epsilon'] = random.uniform(1e-9, 1e-8)
        parameters_mlp['n_iter_no_change'] = random.randint(5, 20)
        parameters_mlp['max_fun'] = random.randint(10000, 20000)
        print("Parameters:", parameters_mlp)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_mlp:\n")
            f.write(f"{parameters_mlp}: {parameters_mlp}\n")
        return parameters_mlp

    def random_search_ridge():
        parameters_ridge = {}
        parameters_ridge['alpha'] = random.uniform(0.1, 1.5)
        parameters_ridge['fit_intercept'] = random.choice([True, False])
        parameters_ridge['normalize'] = random.choice([True, False])
        parameters_ridge['copy_X'] = random.choice([True, False])
        parameters_ridge['max_iter'] = random.randint(10, 100)
        parameters_ridge['tol'] = random.uniform(1e-6, 1e-3)
        parameters_ridge['solver'] = random.choice(['auto', 'svd', 'cholesky', 'sparse_cg', 'lsqr',
'sag', 'saga'])
        parameters_ridge['random_state'] = random.randint(0, 10)
        print("Parameters:", parameters_ridge)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_ridge:\n")
            f.write(f"{parameters_ridge}: {parameters_ridge}\n")
        return parameters_ridge

    def random_search_sgd():
        parameters_sgd = {}
        parameters_sgd['loss'] = random.choice(['squared_loss', 'huber', 'epsilon_insensitive'])
        parameters_sgd['penalty'] = random.choice(['l2', 'l1', 'elasticnet'])
        parameters_sgd['alpha'] = random.uniform(0.0001, 0.001)
        parameters_sgd['l1_ratio'] = random.uniform(0.0, 0.3)
        parameters_sgd['fit_intercept'] = random.choice([True, False])
        parameters_sgd['max_iter'] = random.randint(100, 1000)
        parameters_sgd['tol'] = random.uniform(1e-3, 1e-2)
```

```
        parameters_sgd['shuffle'] = random.choice([True, False])
        parameters_sgd['verbose'] = random.randint(0, 0)
        parameters_sgd['epsilon'] = random.uniform(0.2, 1.0)
        parameters_sgd['random_state'] = random.randint(3, 7)
        parameters_sgd['learning_rate'] = random.choice(['constant', 'optimal', 'invscaling',
'adaptive'])
        parameters_sgd['eta0'] = random.uniform(0.15, 0.35)
        parameters_sgd['power_t'] = random.uniform(0.1, 0.7)
        parameters_sgd['early_stopping'] = random.choice([True, False])
        parameters_sgd['validation_fraction'] = random.uniform(0.01, 1.0)
        parameters_sgd['n_iter_no_change'] = random.randint(1,10)
        parameters_sgd['warm_start'] = random.choice([True, False])
        parameters_sgd['average'] = random.choice([True, False])
        print("Parameters:", parameters_sgd)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_sgd:\n")
            f.write(f"{parameters_sgd}: {parameters_sgd}\n")
        return parameters_sgd

    # Pozivanje nasumicnih parametara

    ARDParam = random_search_ardr()
    BayRidgeParam = random_search_bayridge()
    LarsParam = random_search_lars()
    LassoParam = random_search_lasso()
    LassoLarsParam = random_search_lassolars()
    LinRegParam = random_search_linreg()
    MLPParam = random_search_mlp()
    RidgeParam = random_search_ridge()
    SGDParam = random_search_sgd()

    estimators = [('ARDR', ARDRegression(**ARDParam)),
                  ('BayRidge', BayesianRidge(**BayRidgeParam)),
                  ('Lars', Lars(**LarsParam)),
                  ('Lasso', Lasso(**LassoParam)),
                  ('LassoLars', LassoLars(**LassoLarsParam)),
                  ('LinReg', LinearRegression(**LinRegParam)),
                  ('MLPR', MLPRegressor(**MLPParam)),
                  ('Ridge', Ridge(**RidgeParam)),
                  ('SGDR', SGDRegressor(**SGDParam))
        ]

    model = StackingRegressor(estimators=estimators,
                              final_estimator = None,
                              cv = None)

    cvmodel = cross_validate(model, X_train, y_train, cv=5,
                              scoring = ("r2",
                                         "neg_mean_absolute_error",
                                         "neg_root_mean_squared_error",
                                         "neg_mean_absolute_percentage_error"),
                              return_train_score=True)

    print("All Scores From CV5 = {}".format(cvmodel))
    file1.write("R2 Train Scores = {}\n".format(cvmodel['train_r2']))
```

```
    file1.write("R2 Test Scores = {}\n".format(cvmodel['test_r2']))
    file1.write("MAE Train Scores = {}\n".format(abs(cvmodel['train_neg_mean_absolute_error'])))
    file1.write("MAE Test Scores = {}\n".format(abs(cvmodel['test_neg_mean_absolute_error'])))
    file1.write("RMSE Train Scores = {}\n".format(abs(cvmodel['train_neg_root_mean_squared_error'])))
    file1.write("RMSE Test Scores = {}\n".format(abs(cvmodel['test_neg_root_mean_squared_error'])))
    AvrR2ScoreTrain = np.mean(cvmodel['train_r2'])
    StdR2ScoreTrain = np.std(cvmodel['train_r2'])
    AvrR2ScoreTest = np.mean(cvmodel['test_r2'])
    StdR2ScoreTest = np.std(cvmodel['test_r2'])
    AvrAllR2Score = np.mean([AvrR2ScoreTrain,AvrR2ScoreTest])
    StdAllR2Score = np.std([AvrR2ScoreTrain, AvrR2ScoreTest])
    AvrMAEScoreTrain = np.mean(abs(cvmodel['train_neg_mean_absolute_error']))
    StdMAEScoreTrain = np.std(abs(cvmodel['train_neg_mean_absolute_error']))
    AvrMAEScoreTest = np.mean(abs(cvmodel['test_neg_mean_absolute_error']))
    StdMAEScoreTest = np.std(abs(cvmodel['test_neg_mean_absolute_error']))
    AvrAllMAEScore = np.mean([AvrMAEScoreTrain, AvrMAEScoreTest])
    StdAllMAEScore = np.std([AvrMAEScoreTrain, AvrMAEScoreTest])
    AvrRMSEScoreTrain = np.mean(abs(cvmodel['train_neg_root_mean_squared_error']))
    StdRMSEScoreTrain = np.std(abs(cvmodel['train_neg_root_mean_squared_error']))
    AvrRMSEScoreTest = np.mean(abs(cvmodel['test_neg_root_mean_squared_error']))
    StdRMSEScoreTest = np.std(abs(cvmodel['test_neg_root_mean_squared_error']))
    AvrAllRMSEScore = np.mean([AvrRMSEScoreTrain, AvrRMSEScoreTest])
    StdAllRMSEScore = np.std([AvrRMSEScoreTrain, AvrRMSEScoreTest])
    print("CV-R^2 Score = {}".format(AvrAllR2Score))
    print("CV-STD R^2 Score = {}".format(StdAllR2Score))
    print("CV-MAE Score = {}".format(AvrAllMAEScore))
    print("CV-STD MAE Score = {}".format(StdAllMAEScore))
    print("CV-RMSE Score ={}".format(AvrAllRMSEScore))
    print("CV-STD RMSE Score = {}".format(StdAllRMSEScore))
    file1.write("#################################################################\n"+\
                "AvrR2Score Train = {}\n".format(AvrR2ScoreTrain)+\
                "StdR2Score Train = {}\n".format(StdR2ScoreTrain)+\
                "AvrR2Score Test = {}\n".format(AvrR2ScoreTest)+\
                "StdR2Score Test = {}\n".format(StdR2ScoreTest)+\
                "AvrAllR2Score = {}\n".format(AvrAllR2Score)+\
                "StdAllR2Score = {}\n".format(StdAllR2Score)+\
                "AvrMAEScore Train = {}\n".format(AvrMAEScoreTrain)+\
                "StdMAEScore Train = {}\n".format(StdMAEScoreTrain)+\
                "AvrMAEScore Test = {}\n".format(AvrMAEScoreTest)+\
                "StdMAEScore Test ={}\n".format(StdMAEScoreTest)+\
                "AvrAllMAEScore = {}\n".format(AvrAllMAEScore)+\
                "StdAllMAEScore = {}\n".format(StdAllMAEScore)+\
                "AvrRMSEScore Train = {}\n".format(AvrRMSEScoreTrain)+\
                "StdRMSEScore Train = {}\n".format(StdRMSEScoreTrain)+\
                "AvrRMSEScore Test = {}\n".format(AvrRMSEScoreTest)+\
                "StdRMSEScore Test = {}\n".format(StdRMSEScoreTest)+\
                "AvrAllRMSEScore = {}\n".format(AvrAllRMSEScore)+\
                "StdAllRMSEScore = {}\n".format(StdAllRMSEScore)+\
                "#################################################################\n")

    if AvrAllR2Score > 0.99:
            print("#################################################################")
            print(" Final Evaluation")
            print("#################################################################")
            file1.write("#################################################################\n")
```

```
            file1.write(" Final Evaluation\n")
            file1.write("############################################################\n")
            model.fit(X_train,y_train)
            R2Test = model.score(X_test,y_test)
            MAETest = mean_absolute_error(y_test, model.predict(X_test))
            RMSETest = np.sqrt(mean_squared_error(y_test, model.predict(X_test)))
            print("R^2 Test = {}".format(R2Test))
            print("MAE Test = {}".format(MAETest))
            print("RMSE Test = {}".format(RMSETest))
            file1.write("############################################################\n")
            file1.write("Final R^2 Test = {}\n".format(R2Test))
            file1.write("Final MAE Test = {}\n".format(MAETest))
            file1.write("Final RMSE Test = {}\n".format(RMSETest))
            file1.write("############################################################\n")
            file1.flush()
            return R2Test
    else:
            return AvrAllR2Score

name = "StackingY1"
file1 = open("{}_StackingY1_Score.dat".format(name),"w")

while True:
    res = StackingEnsemble(X_train, X_test, y_train, y_test)
    if res > 0.99:
        print("Solution is Found!")
        break
    else:
        continue

file1.close()
```

DODATAK G

```
# Voting ansambl, Y1

# Umetanje potrebnih knjiznica
from sklearn.ensemble import VotingRegressor
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import random
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_validate

# Umetanje potrebnih algoritama
from sklearn.linear_model import ARDRegression
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import Lars
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoLars
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import SGDRegressor

path = r"C:\Users\tomev\Desktop\dataset.csv"
data_a = pd.read_csv(path) # Dataset za gadanje Y1
data_b = pd.read_csv(path) # Dataset za gadanje Y1

# Gadamo Y1
print('Gadamo Y1')
data2_a = data_a.pop('Y2') # Izvucen Y2
y = data_a.pop('Y1') # Izvucen Y1
X = data_a # Definiran X

scaler = StandardScaler()
X = scaler.fit_transform(X) # Skaliranje ulaznih varijabli

file_path_param_stackingY2 = r"C:\Users\tomev\Desktop\Stacking_Rezultati\Voting_HiperparametriY1.txt"

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

def StackingEnsemble (X_train, X_test, y_train, y_test):

    # Nasumicni odabir parametara

    def random_search_ardr():
        parameters_ardr = {}
        parameters_ardr['n_iter'] = random.randint(250, 350)
        parameters_ardr['tol'] = random.uniform(1e-5, 1e-3)
        parameters_ardr['alpha_1'] = random.uniform(1e-2, 1e-5)
        parameters_ardr['alpha_2'] = random.uniform(1e-8, 1e-5)
        parameters_ardr['lambda_1'] = random.uniform(1e-7, 1e-6)
        parameters_ardr['lambda_2'] = random.uniform(1e-6, 1e-5)
```

```
        parameters_ardr['compute_score'] = random.choice([True, False])
        parameters_ardr['threshold_lambda'] = random.uniform(9000, 11000)
        parameters_ardr['fit_intercept'] = random.choice([True, False])
        parameters_ardr['normalize'] = random.choice([True, False])
        parameters_ardr['copy_X'] = random.choice([True, False])
        parameters_ardr['verbose'] = random.choice([True, False])
        print("Parameters:", parameters_ardr)
        with open(file_path_param_stackingY2, "w") as f:
            f.write("parameters_ardr:\n")
            f.write(f"{parameters_ardr}: {parameters_ardr}\n")
        return parameters_ardr

    def random_search_bayridge():
        parameters_bayridge = {}
        parameters_bayridge['n_iter'] = random.randint(300, 400)
        parameters_bayridge['tol'] = random.uniform(1e-4, 1e-2)
        parameters_bayridge['alpha_1'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['alpha_2'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['lambda_1'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['lambda_2'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['alpha_init'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['lambda_init'] = random.uniform(1e-7, 1e-5)
        parameters_bayridge['compute_score'] = random.choice([True, False])
        parameters_bayridge['fit_intercept'] = random.choice([True, False])
        parameters_bayridge['normalize'] = random.choice([True, False])
        parameters_bayridge['copy_X'] = random.choice([True, False])
        parameters_bayridge['verbose'] = random.choice([True, False])
        print("Parameters:", parameters_bayridge)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_bayridge:\n")
            f.write(f"{parameters_bayridge}: {parameters_bayridge}\n")
        return parameters_bayridge

    def random_search_lars():
        parameters_lars = {}
        parameters_lars['fit_intercept'] = random.choice([True, False])
        parameters_lars['verbose'] = random.choice([True, False])
        parameters_lars['normalize'] = random.choice([True, False])
        parameters_lars['precompute'] = random.choice(['auto', True, False])
        parameters_lars['n_nonzero_coefs'] = random.randint(10, 100)
        parameters_lars['eps'] = random.uniform(1e-16, 1e-10)
        parameters_lars['copy_X'] = random.choice([True, False])
        parameters_lars['fit_path'] = random.choice([True, False])
        parameters_lars['jitter'] = random.uniform(1e-10, 1e-4)
        parameters_lars['random_state'] = random.randint(1, 100)
        print("Parameters:", parameters_lars)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_lars:\n")
            f.write(f"{parameters_lars}: {parameters_lars}\n")
        return parameters_lars

    def random_search_lasso():
        parameters_lasso = {}
        parameters_lasso['alpha'] = random.uniform(0.01, 1.0)
        parameters_lasso['fit_intercept'] = random.choice([True, False])
```

```
        parameters_lasso['normalize'] = random.choice([True, False])
        parameters_lasso['precompute'] = random.choice([True, False])
        parameters_lasso['copy_X'] = random.choice([True, False])
        parameters_lasso['max_iter'] = random.randint(100, 1000)
        parameters_lasso['tol'] = random.uniform(1e-6, 1e-3)
        parameters_lasso['warm_start'] = random.choice([True, False])
        parameters_lasso['positive'] = random.choice([True, False])
        parameters_lasso['random_state'] = random.randint(1, 100)
        parameters_lasso['selection'] = random.choice(['cyclic', 'random'])
        print("Parameters:", parameters_lasso)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_lasso:\n")
            f.write(f"{parameters_lasso}: {parameters_lasso}\n")
        return parameters_lasso

    def random_search_lassolars():
        parameters_lassolars = {}
        parameters_lassolars['alpha'] = random.uniform(0.001, 0.1)
        parameters_lassolars['fit_intercept'] = random.choice([True, False])
        parameters_lassolars['verbose'] = random.choice([True, False])
        parameters_lassolars['normalize'] = random.choice([True, False])
        parameters_lassolars['precompute'] = random.choice([True, False, 'auto'])
        parameters_lassolars['max_iter'] = random.randint(100, 1000)
        parameters_lassolars['eps'] = random.uniform(np.finfo(float).eps, 1e-3)
        parameters_lassolars['copy_X'] = random.choice([True, False])
        parameters_lassolars['fit_path'] = random.choice([True, False])
        parameters_lassolars['positive'] = random.choice([True, False])
        parameters_lassolars['jitter'] = random.uniform(0.01, 0.1)
        parameters_lassolars['random_state'] = random.randint(1, 100)
        print("Parameters:", parameters_lassolars)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_lassolars:\n")
            f.write(f"{parameters_lassolars}: {parameters_lassolars}\n")
        return parameters_lassolars

    def random_search_linreg():
        parameters_linreg = {
        'fit_intercept': random.choice([True, False]),
        'normalize': random.choice([True, False]),
        'copy_X': random.choice([True, False]),
        'n_jobs': random.randint(1, 100),
        'positive': random.choice([True, False])
        }
        print("Parameters:", parameters_linreg)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_linreg:\n")
            f.write(f"{parameters_linreg}: {parameters_linreg}\n")
        return parameters_linreg

    def random_search_mlp():
        parameters_mlp = {}
        numHidLayers = random.randint(2, 10)
        hidLayers = []
        for i in range(1, numHidLayers, 1):
            hidLayers.append(random.randint(10, 100))
```

```
        tupleHid = tuple(hidLayers)
        parameters_mlp['hidden_layer_sizes'] = tupleHid
        parameters_mlp['activation'] = random.choice(['identity', 'logistic', 'tanh', 'relu'])
        parameters_mlp['solver'] = random.choice(['lbfgs', 'sgd', 'adam'])
        parameters_mlp['alpha'] = random.uniform(1e-5, 1e-3)
        parameters_mlp['batch_size'] = random.randint(1, min(X_train.shape[0], 200))
        parameters_mlp['learning_rate'] = random.choice(['constant', 'invscaling', 'adaptive'])
        parameters_mlp['learning_rate_init'] = random.uniform(1e-5, 1e-4)
        parameters_mlp['power_t'] = random.uniform(0.3, 0.7)
        parameters_mlp['max_iter'] = random.randint(300, 500)
        parameters_mlp['shuffle'] = random.choice([True, False])
        parameters_mlp['random_state'] = random.randint(0, 150)
        parameters_mlp['tol'] = random.uniform(1e-5, 1e-3)
        parameters_mlp['verbose'] = random.choice([True, False])
        parameters_mlp['warm_start'] = random.choice([True, False])
        parameters_mlp['momentum'] = random.uniform(0.2, 0.8)
        parameters_mlp['nesterovs_momentum'] = random.choice([True, False])
        parameters_mlp['early_stopping'] = random.choice([True, False])
        parameters_mlp['validation_fraction'] = random.uniform(0.1, 0.3)
        parameters_mlp['beta_1'] = random.uniform(0.7, 0.9)
        parameters_mlp['beta_2'] = random.uniform(0.8, 0.99)
        parameters_mlp['epsilon'] = random.uniform(1e-9, 1e-8)
        parameters_mlp['n_iter_no_change'] = random.randint(5, 20)
        parameters_mlp['max_fun'] = random.randint(10000, 20000)
        print("Parameters:", parameters_mlp)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_mlp:\n")
            f.write(f"{parameters_mlp}: {parameters_mlp}\n")
        return parameters_mlp

    def random_search_ridge():
        parameters_ridge = {}
        parameters_ridge['alpha'] = random.uniform(0.1, 1.5)
        parameters_ridge['fit_intercept'] = random.choice([True, False])
        parameters_ridge['normalize'] = random.choice([True, False])
        parameters_ridge['copy_X'] = random.choice([True, False])
        parameters_ridge['max_iter'] = random.randint(10, 100)
        parameters_ridge['tol'] = random.uniform(1e-6, 1e-3)
        parameters_ridge['solver'] = random.choice(['auto', 'svd', 'cholesky', 'sparse_cg', 'lsqr',
'sag', 'saga'])
        parameters_ridge['random_state'] = random.randint(0, 10)
        print("Parameters:", parameters_ridge)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_ridge:\n")
            f.write(f"{parameters_ridge}: {parameters_ridge}\n")
        return parameters_ridge

    def random_search_sgd():
        parameters_sgd = {}
        parameters_sgd['loss'] = random.choice(['squared_loss', 'huber', 'epsilon_insensitive'])
        parameters_sgd['penalty'] = random.choice(['l2', 'l1', 'elasticnet'])
        parameters_sgd['alpha'] = random.uniform(0.0001, 0.001)
        parameters_sgd['l1_ratio'] = random.uniform(0.0, 0.3)
        parameters_sgd['fit_intercept'] = random.choice([True, False])
        parameters_sgd['max_iter'] = random.randint(100, 1000)
```

```
        parameters_sgd['tol'] = random.uniform(1e-3, 1e-2)
        parameters_sgd['shuffle'] = random.choice([True, False])
        parameters_sgd['verbose'] = random.randint(0, 0)
        parameters_sgd['epsilon'] = random.uniform(0.2, 1.0)
        parameters_sgd['random_state'] = random.randint(3, 7)
        parameters_sgd['learning_rate'] = random.choice(['constant', 'optimal', 'invscaling',
'adaptive'])
        parameters_sgd['eta0'] = random.uniform(0.15, 0.35)
        parameters_sgd['power_t'] = random.uniform(0.1, 0.7)
        parameters_sgd['early_stopping'] = random.choice([True, False])
        parameters_sgd['validation_fraction'] = random.uniform(0.01, 1.0)
        parameters_sgd['n_iter_no_change'] = random.randint(1,10)
        parameters_sgd['warm_start'] = random.choice([True, False])
        parameters_sgd['average'] = random.choice([True, False])
        print("Parameters:", parameters_sgd)
        with open(file_path_param_stackingY2, "a") as f:
            f.write("parameters_sgd:\n")
            f.write(f"{parameters_sgd}: {parameters_sgd}\n")
        return parameters_sgd

    # Pozivanje nasumicnih parametara

    ARDParam = random_search_ardr()
    BayRidgeParam = random_search_bayridge()
    LarsParam = random_search_lars()
    LassoParam = random_search_lasso()
    LassoLarsParam = random_search_lassolars()
    LinRegParam = random_search_linreg()
    MLPParam = random_search_mlp()
    RidgeParam = random_search_ridge()
    SGDParam = random_search_sgd()

    estimators = [('ARDR', ARDRegression(**ARDParam)),
                  ('BayRidge', BayesianRidge(**BayRidgeParam)),
                  ('Lars', Lars(**LarsParam)),
                  ('Lasso', Lasso(**LassoParam)),
                  ('LassoLars', LassoLars(**LassoLarsParam)),
                  ('LinReg', LinearRegression(**LinRegParam)),
                  ('MLPR', MLPRegressor(**MLPParam)),
                  ('Ridge', Ridge(**RidgeParam)),
                  ('SGDR', SGDRegressor(**SGDParam))
        ]

    model = VotingRegressor(estimators=estimators)

    cvmodel = cross_validate(model, X_train, y_train, cv=5,
                             scoring = ("r2",
                                        "neg_mean_absolute_error",
                                        "neg_root_mean_squared_error",
                                        "neg_mean_absolute_percentage_error"),
                             return_train_score=True)

    print("All Scores From CV5 = {}".format(cvmodel))
    file1.write("R2 Train Scores = {}\n".format(cvmodel['train_r2']))
    file1.write("R2 Test Scores = {}\n".format(cvmodel['test_r2']))
```

```
    file1.write("MAE Train Scores = {}\n".format(abs(cvmodel['train_neg_mean_absolute_error'])))
    file1.write("MAE Test Scores = {}\n".format(abs(cvmodel['test_neg_mean_absolute_error'])))
    file1.write("RMSE Train Scores = {}\n".format(abs(cvmodel['train_neg_root_mean_squared_error'])))
    file1.write("RMSE Test Scores = {}\n".format(abs(cvmodel['test_neg_root_mean_squared_error'])))
    AvrR2ScoreTrain = np.mean(cvmodel['train_r2'])
    StdR2ScoreTrain = np.std(cvmodel['train_r2'])
    AvrR2ScoreTest = np.mean(cvmodel['test_r2'])
    StdR2ScoreTest = np.std(cvmodel['test_r2'])
    AvrAllR2Score = np.mean([AvrR2ScoreTrain,AvrR2ScoreTest])
    StdAllR2Score = np.std([AvrR2ScoreTrain, AvrR2ScoreTest])
    AvrMAEScoreTrain = np.mean(abs(cvmodel['train_neg_mean_absolute_error']))
    StdMAEScoreTrain = np.std(abs(cvmodel['train_neg_mean_absolute_error']))
    AvrMAEScoreTest = np.mean(abs(cvmodel['test_neg_mean_absolute_error']))
    StdMAEScoreTest = np.std(abs(cvmodel['test_neg_mean_absolute_error']))
    AvrAllMAEScore = np.mean([AvrMAEScoreTrain, AvrMAEScoreTest])
    StdAllMAEScore = np.std([AvrMAEScoreTrain, AvrMAEScoreTest])
    AvrRMSEScoreTrain = np.mean(abs(cvmodel['train_neg_root_mean_squared_error']))
    StdRMSEScoreTrain = np.std(abs(cvmodel['train_neg_root_mean_squared_error']))
    AvrRMSEScoreTest = np.mean(abs(cvmodel['test_neg_root_mean_squared_error']))
    StdRMSEScoreTest = np.std(abs(cvmodel['test_neg_root_mean_squared_error']))
    AvrAllRMSEScore = np.mean([AvrRMSEScoreTrain, AvrRMSEScoreTest])
    StdAllRMSEScore = np.std([AvrRMSEScoreTrain, AvrRMSEScoreTest])

    print("CV-R^2 Score = {}".format(AvrAllR2Score))
    print("CV-STD R^2 Score = {}".format(StdAllR2Score))
    print("CV-MAE Score = {}".format(AvrAllMAEScore))
    print("CV-STD MAE Score = {}".format(StdAllMAEScore))
    print("CV-RMSE Score ={}".format(AvrAllRMSEScore))
    print("CV-STD RMSE Score = {}".format(StdAllRMSEScore))
    file1.write("#################################################################\n"+\
                "AvrR2Score Train = {}\n".format(AvrR2ScoreTrain)+\
                "StdR2Score Train = {}\n".format(StdR2ScoreTrain)+\
                "AvrR2Score Test = {}\n".format(AvrR2ScoreTest)+\
                "StdR2Score Test = {}\n".format(StdR2ScoreTest)+\
                "AvrAllR2Score = {}\n".format(AvrAllR2Score)+\
                "StdAllR2Score = {}\n".format(StdAllR2Score)+\
                "AvrMAEScore Train = {}\n".format(AvrMAEScoreTrain)+\
                "StdMAEScore Train = {}\n".format(StdMAEScoreTrain)+\
                "AvrMAEScore Test = {}\n".format(AvrMAEScoreTest)+\
                "StdMAEScore Test ={}\n".format(StdMAEScoreTest)+\
                "AvrAllMAEScore = {}\n".format(AvrAllMAEScore)+\
                "StdAllMAEScore = {}\n".format(StdAllMAEScore)+\
                "AvrRMSEScore Train = {}\n".format(AvrRMSEScoreTrain)+\
                "StdRMSEScore Train = {}\n".format(StdRMSEScoreTrain)+\
                "AvrRMSEScore Test = {}\n".format(AvrRMSEScoreTest)+\
                "StdRMSEScore Test = {}\n".format(StdRMSEScoreTest)+\
                "AvrAllRMSEScore = {}\n".format(AvrAllRMSEScore)+\
                "StdAllRMSEScore = {}\n".format(StdAllRMSEScore)+\
                "#################################################################\n")

    if AvrAllR2Score > 0.99:
            print("#################################################################")
            print(" Final Evaluation")
            print("#################################################################")
            file1.write("#################################################################\n")
```

```
            file1.write(" Final Evaluation\n")
            file1.write("##############################################################\n")
            model.fit(X_train,y_train)
            R2Test = model.score(X_test,y_test)
            MAETest = mean_absolute_error(y_test, model.predict(X_test))
            RMSETest = np.sqrt(mean_squared_error(y_test, model.predict(X_test)))
            print("R^2 Test = {}".format(R2Test))
            print("MAE Test = {}".format(MAETest))
            print("RMSE Test = {}".format(RMSETest))
            file1.write("##############################################################\n")
            file1.write("Final R^2 Test = {}\n".format(R2Test))
            file1.write("Final MAE Test = {}\n".format(MAETest))
            file1.write("Final RMSE Test = {}\n".format(RMSETest))
            file1.write("##############################################################\n")
            file1.flush()
            return R2Test
    else:
            return AvrAllR2Score

name = "VotingY1"
file1 = open("{}_VotingY1_Score.dat".format(name),"w")

while True:
    res = StackingEnsemble(X_train, X_test, y_train, y_test)
    if res > 0.99:
        print("Solution is Found!")
        break
    else:
        continue

file1.close()
```