Dijagnostika elektromotora modelima umjetne inteligencije

Ladić, Tin

Master's thesis / Diplomski rad

2022

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:557748

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

DIJAGNOSTIKA ELEKTROMOTORA MODELIMA UMJETNE INTELIGENCIJE

Rijeka, rujan 2022.

Tin Ladić

0069066453

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

DIJAGNOSTIKA ELEKTROMOTORA MODELIMA UMJETNE INTELIGENCIJE

Mentor: Prof. dr. sc. Zlatan Car

Rijeka, rujan 2022.

Tin Ladić

0069066453

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 21. ožujka 2022.

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

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Tin Ladić (0069066453)**
Studij: Diplomski sveučilišni studij elektrotehnike
Modul: Automatika

Zadatak: **Dijagnostika elektromotora modelima umjetne inteligencije/Electromotor diagnostics using artificial intelligence methods**

Opis zadatka:

Izraditi pregled literature u području dijagnostike strojeva primjenom umjetne inteligencije. Primijeniti algoritme umjetne inteligencije na set podataka. Predobraditi set podataka korištenjem algoritama za predobradu, te ponoviti modeliranje. Usporediti performanse algoritama umjetne inteligencija sa i bez predobrade, te komentirati utjecaj hiperparametara.

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

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Prof. dr. sc. Zlatan Car

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Viktor Sučić

IZJAVA

Sukladno članku 9. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci izjavljujem da sam izradio ovaj diplomski rad samostalno, koristeći vlastito znanje i navedenu literaturu, u razdoblju od zadavanja zadatka do datuma predaje.

Jan Lozić

Ime i prezime studenta

ZAHVALA

Zahvaljujem prof. dr. sc. Zlatanu Caru na prihvaćanju mentorstva za izradu rada i asist. Sandiju Baressi Šegoti, mag. ing. comp. na uputama i pomoći tijekom izrade.

Sadržaj

1. UVOD

U ovom diplomskom radu obraditi će se problematika dijagnostike elektromotornih pogona metodama umjetne inteligencije. Nakon kratkog pregleda postojeće literature na ovu temu opisati će se konkretni dataset koji će se koristiti za klasifikaciju u ovome radu, otkuda je preuzet, kako su prikupljeni podaci i kako su predobrađeni. Zatim će se opisati daljnje metode obrade podataka, konkretno analiza glavnih komponenti i uklanjanje jako koreliranih podataka. Nakon pripreme podataka potrebno je odabrati algoritam strojnog učenja koji će se koristiti. Biti će opisani algoritmi koji se koriste u ovom radu a to su *MLPClassifier* i *XGBClassifier*. Također opisati će se način odabira i optimizacije hiperparametara. Na kraju će se primijeniti opisani algoritmi te će se komentirati rezultati i utjecaj metoda obrade podataka na pojedini algoritam.

2. DIJAGNOSTIKA ELEKTROMOTORNIH POGONA PRIMJENOM UMJETNE INTELIGENCIJE

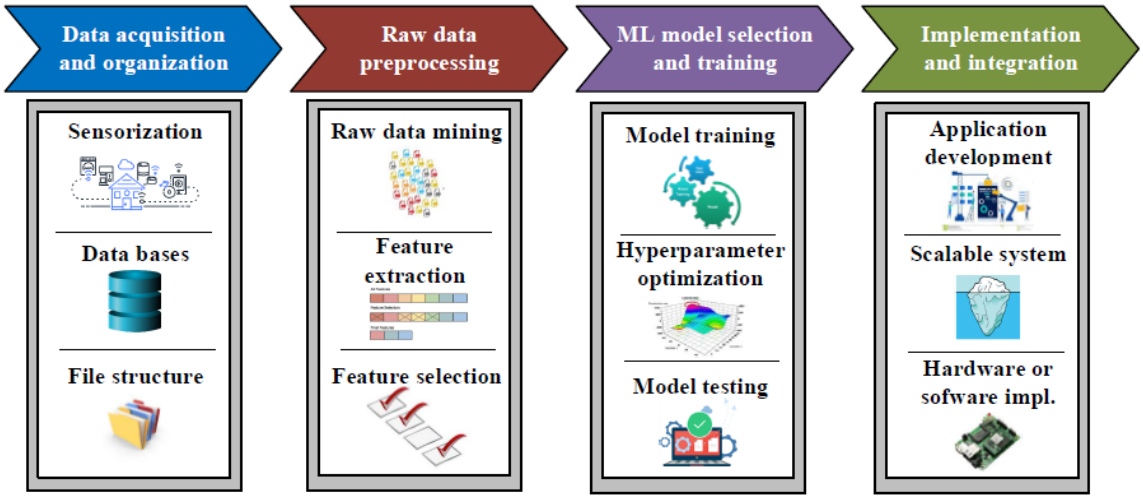
U današnjoj industriji javlja se potreba za konstantnim praćenjem stanja elektromotornih pogona radi detektiranja potencijalnih kvarova u nastajanju i sprječavanja istih. Za praćenje stanja stroja u pogonu tradicionalno su se koristile metode bazirane na matematičkom modelu stroja i metode bazirane na praćenju signala dobivenih sa stvarnog stroja [1]. Ove metode su ugrađivane u sami upravljački softver elektromotornih pogona. U današnje vrijeme se metode za praćenje stanja stroja ne ugrađuju u samu upravljačku elektroniku stroja, već se nude kao on-line usluga [1]. Ove usluge temelje se na prikupljanju velikih količina podataka o radu stroja i spremanju u baze podataka za daljnju obradu. U tablici 2.1. prikazani su neki od sustava za praćenje stanja stroja u radu.

Tablica 2.1. On-line sustavi praćenja stanja stroja [1]

Proizvođač	**Područje rada**	**Naziv sustava**	**Online platforma**
Alstom	Željeznica	HealthHub	Google Cloud
Bombardier	Željeznica	Optiflo	IBM Cloud
Siemens Mobility	Željeznica	Railigent	AWS
Hitachi	Željeznica	Lumada	Hitachi Smart Cloud
CAF	Željeznica	LeadMind	AWS
KONE	Vertikalni transport (dizala)	KONE CARE	IBM Cloud
Thyssenkrupp Elevator	Vertikalni transport (dizala)	MAX	Azure Cloud
Otis	Vertikalni transport (dizala)	Otis ONE	Azure Cloud
Siemens Gamesa	Proizvodnja energije (Vjetroelektrane)	Pythia	-
Vestas	Proizvodnja energije (Vjetroelektrane)	-	TIBCO Spotfire

Umjetna inteligencija u industriji još nije zaživjela kao u komercijalnim primjenama. Problem je u tome što kod komercijalnih primjena podatke koje algoritmi umjetne inteligencije obrađuju proizvode ljudi te ti podaci dosta dobro opisuju problem koji se obrađuje, s druge strane u industriji se podaci prikupljaju iz senzora te ti podaci mogu sadržavati puno nebitnih informacija poput šuma i slično, pa je takve podatke potrebno dodatno obraditi i izdvojiti korisne informacije iz njih. Još jedan problem u industriji je to što nema puno podataka o neispravnom radu stroja, zbog toga što su sustavi dizajnirani da se što manje kvare i kvarovi i abnormalne situacije su rijetke [1].

Općeniti je dojam da za je razvijanje algoritma strojnog učenja potrebno samo odabrati i optimizirati algoritam, no u stvarnosti je to puno složeniji proces, koji je prikazan na slici 2.1.

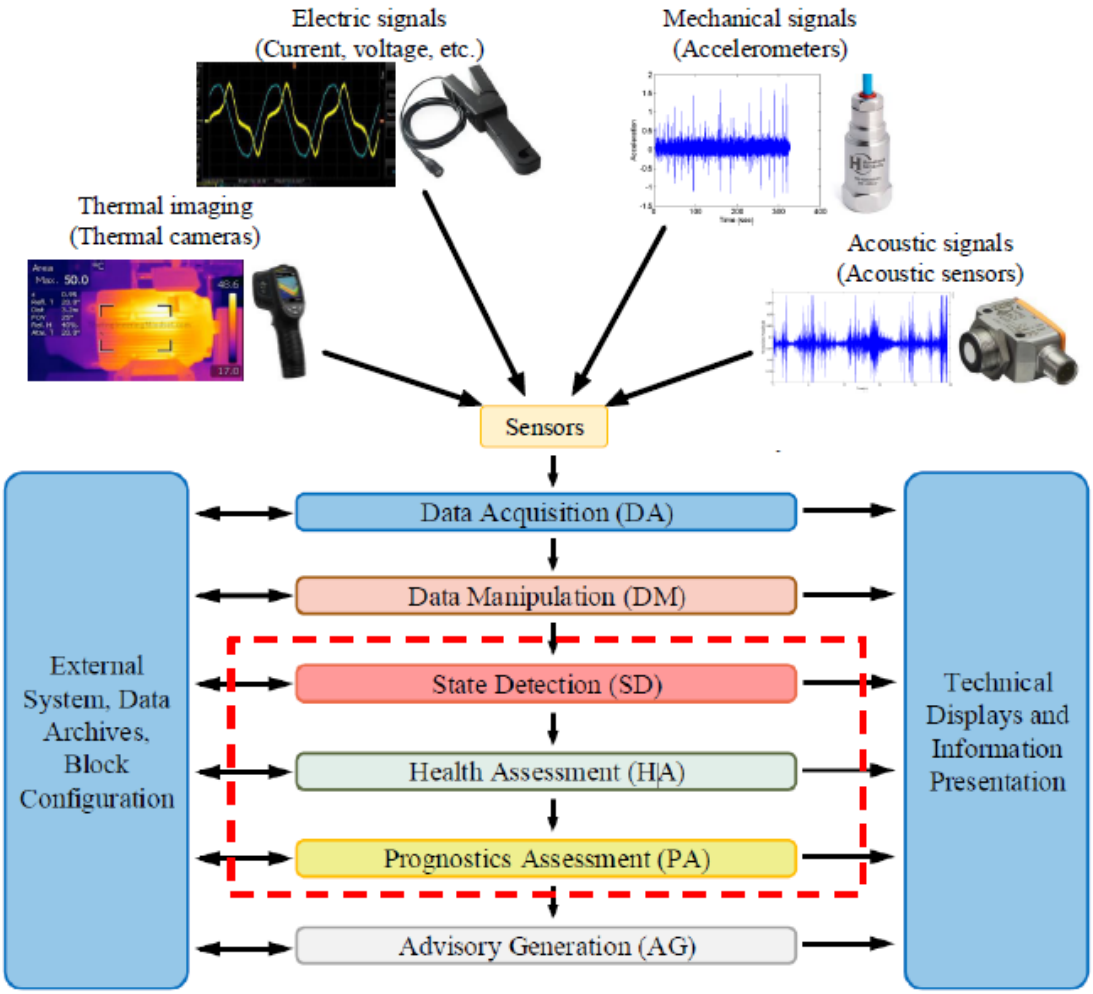


Slika 2.1. Dijagram toka razvijanja algoritma za strojno učenje [1]

Na početku je potrebno odabrati ili saznati koja metodologija se koristi za nadzor rada stroja, zatim je potrebno prikupiti i rasporediti podatke (*eng. Data acquisition and organisation*) i nakon toga te iste podatke obraditi i iz njih izvući značajke koje su korisne za detekciju i dijagnostiku elektromotornog pogona (*eng. Raw data preproccesing*) [1]. Nakon pripreme podataka potrebno je odabrati model, optimizirati hiperparametre, trenirati ga i testirati ima li zadovoljavajuće rezultate (*eng. ML model selection and training*), razvijeni model je nakon toga potrebno implementirati ili integrirati na postojećem stvarnom sustavu (*eng. Implementation and integration*) [1]. Svaki od ovih koraka biti će opisan u narednim poglavljima.

2.1. Metodologije nadzora rada stroja

Trenutno postoji više standarda za prikupljanje informacija i nadzor rada strojeva, poput ISO-13374 i CRISP-DM. Blok dijagram ISO-13374 standarda vidljiv je na slici 2.2.



Slika 2.2. Blok dijagram ISO-13374 standarda za nadzor rada stroja [1]

Svaki algoritam nadzora sadrži tri glavna bloka, blok za prikupljanje podataka (*eng. Data Acquisition*), blok za manipulaciju podacima (*eng. Data Manipulation*) i blok za prijavu i interpretaciju greške (*eng. Advisory Generation*) [1]. Blokovi označeni crveno iscrtkanim pravokutnikom na slici 2.1. mogu a i ne moraju biti dio algoritma, ovisno o metodologiji nadzora koja je odabrana.

Postoje četiri glavne metodologije nadzora:

- Metodologija zasnovana na praćenju signala stroja,
- Metodologija zasnovana na korištenju matematičkog modela stroja,
- Metodologija zasnovana na prikupljanju podataka,
- Hibridna metodologija [1].

Metodologija zasnovana na praćenju signala stroja prati signale sa stvarnog stroja (struja, napon, vibracije, itd.) i uspoređuje ih sa signalima koji predstavljaju normalan rad stroja. Ukoliko postoji odstupanje među tim signalima moguće je da postoji kvar na stroju i to se prijavljuje upravljačkoj elektronici i na kraju ljudskom operateru [1]. Ovom metodom moguće je detektirati i dijagnosticirati kvar.

Metodologija zasnovana na korištenju matematičkog modela stroja uspoređuje stvarni stroj s matematičkim modelom u stvarnom vremenu, pritom koristeći iste ulazne varijable koje su izmjerene na stvarnom stroju [1]. Izlazi iz stroja i iz modela se uspoređuju i ovisno o njihovoj razlici se utvrđuje kvar. Ovom metodom moguće je dijagnosticirati kvar.

Metodologija prikupljanja podataka koristi velike baze podataka o radu stroja i trenira algoritme umjetne inteligencije koji detektiraju skrivene uzorke u podacima, u tako trenirane algoritme ubacuju se podaci iz stvarnog vremena i algoritam detektira i dijagnosticira postojeće kvarove, te predviđa buduće kvarove na stroju [1].

Hibridna metodologija koristi više navedenih metodologija zajedno kako bi što točnije nadzirala rad stroja.

2.2. Prikupljanje podataka

Prikupljanje podataka se vrši sa svih senzora ugrađenih na stroj, prikupljaju se podaci o ulaznim i izlaznim signalima sa stroja poput struje, napona i izlaznog okretnog momenta. Osim prikupljanja podataka u stvarnom vremenu gledaju se podaci i iz baza podataka o istim ili sličnim strojevima koje se nalaze na lokalnim ili udaljenim serverima. Bitna je raznolikost podataka u smislu da su obuhvaćene sve moguće radne točke stroja, i da je obuhvaćen i ispravni i neispravni rad stroja. U industriji postoji nedostatak podataka za neispravna stanja strojeva kao što je već navedeno, zbog toga podaci o strojevima nisu pravilno balansirani te je

potrebno na neki način pribaviti više podataka o neispravnom radu stroja. Jedan od načina je proširivanje dostupnih podataka podacima iz simulacija nekog stroja [1].

2.3. Obrada podataka

Podatke je prije unošenja u algoritam strojnog učenja potrebno obraditi i iz podataka izvući značajke. Obradi podataka pristupa se tako da se prvo podaci „čiste“, to podrazumijeva filtriranje smetnji i detektiranje i odbacivanje pogrešnih mjerenja i nepotpunih podataka. Nakon čišćenja se najčešće normaliziraju podaci, ili čak segmentiraju [1]. Čišćenjem podataka smanjuje se ukupni volumen podataka, te je iz takvih podataka lakše izvući značajke koje će se koristiti prilikom strojnog učenja. Prilikom obrade podataka cilj je transformiranje početnih podataka u numeričke vrijednosti koje se mogu upotrebljavati u algoritmima strojnog učenja, a da se pritom zadrži informacija iz originalnih podataka. Te numeričke vrijednosti zovu se značajke. Značajke se izvlače iz originalnih podataka zbog toga što algoritmi strojnog učenja najčešće ne funkcioniraju najbolje s originalnim oblikom podataka. Značajke se mogu izvlačiti na puno načina, najčešće se koriste transformacije iz vremenske domene u frekvencijsku i obratno, brza Fourierova transformacija (FFT), valićna transformacija i Fourierova transformacija na vremenskom otvoru (STFT). Postoje i razni algoritmi za automatsku transformaciju podataka, poput Analize glavnih komponenti (PCA), Analize linearne diskriminante (LDA) ili Analize nezavisnih komponenti (ICA). Obično se ovim metodama izdvajanja značajki dobije jako velik broj značajki, ali ne sadrže sve značajke tražene informacije o nekom stroju, i neke značajke su redundantne. Potrebno je ukloniti beznačajne i redundantne značajke te tako smanjiti ukupni broj značajki za korištenje prilikom strojnog učenja, time se ubrzava treniranje algoritma strojnog učenja, povećava se točnost i smanjuje se mogućnost *overfittinga* [1]. *Overfitting* predstavlja slučaj kada se neki algoritam previše prilagodio podacima s kojima je treniran i za rezultat ima malu točnost kod testiranja s nekim drugim podacima.

2.4. Odabir i treniranje modela strojnog učenja

Nakon odabira prikladnih značajki potrebno je odabrati, trenirati i optimizirati algoritam strojnog učenja. Postoji mnogo algoritama za strojno učenje no najraširenije su umjetne neuronske mreže (*eng. ANN-Artificial neural networks*). Uz neuronske mreže postoje i konvoluirane neuronske mreže (*eng. CNN-Convoluted Neural Networks*), metoda potpornih vektora (*eng. SVM-Support Vector Method*), metoda K-najbližih susjeda (*eng. KNN – K-Nearest Neighbour*) i genetski algoritmi. Nakon odabira odgovarajućeg modela potrebno je odabrati metodu učenja algoritma. Postoje tri metode učenja, koje su navedene na slici 2.3.

Slika 2.3. Metode učenja algoritma [2]

Učenje sa učiteljem podrazumijeva da algoritam ima dostupne i ulazne i izlazne vrijednosti za treniranje, algoritam tada obrađuje ulazne vrijednosti i uspoređuje rezultate sa poznatim izlaznim vrijednostima [2]. Pogreška se računa kao razlika između dobivenog i željenog rezultata, ta informacija se vraća na početak kako bi se algoritam prilagodio da bi bio točniji. Ovaj proces se ponavlja više puta sa istim setom podataka koji se naziva set za treniranje, treniranje se ponavlja sve dok se ne postigne maksimalna moguća točnost algoritma ili dok se ne pređe broj maksimalnih dozvoljenih iteracija.

Kod učenja bez učitelja algoritam ima samo ulazne vrijednosti i ne zna se kako bi trebale izgledati izlazne vrijednosti. Kod ovakvog učenja treniranje se svodi na grupiranje podataka prema sličnosti.

Kod Ojačanog učenja (*eng. Reinforcement learning*) algoritam obrađuje ulazne podatke, i nakon toga dobiva jednu logičku ili realnu vrijednost koja označava da li je rezultat točan ili netočan [2].

Nakon treniranja algoritma odabranom metodom učenja potrebno je testirati algoritam s drugim setom podataka. Ukoliko rezultati nisu zadovoljavajući potrebno je optimizirati hiperparametre algoritma kako bi se dobio što bolji rezultat.

2.5. Implementacija modela strojnog učenja

Trenirani i testirani modeli se trebaju prilagoditi za implementaciju u industriji. Modeli se najčešće razvijaju pomoću akademskih alata kao što je Matlab, tako da je razvijene modele potrebno prilagoditi za primjenu na odgovarajućoj platformi koja se koristi u industriji [1]. Također je potrebno razviti adekvatne vizualizacije za prikaz rezultata modela strojnog učenja.

Glavni cilj dijagnostike strojeva je informiranje ljudskog operatera o stanju i mogućim problemima u radu stroja. Danas se razvijaju sustavi potpore odlučivanja (*eng. decision support*) koji nastoje osigurati veliki broj informacija o trenutnom stanju stroja ljudskom operateru koji onda odlučuje hoće li i koje korake će poduzeti [3]. Osim detektiranja neispravnog stanja stroja, važno je operateru predočiti što je sustav praćenja navelo na taj zaključak [3]. Sustavi se mogu podijeliti po dva karakteristična načina „objašnjavanja odluke“, na bijele i crne kutije (*eng. white box i black box*). Bijela kutija predstavlja jednostavan sustav klasifikacije poput jednostrukog stabla odluke ili skupine pravila po kojima se vrši detekcija stanja/kvara. Ovakve sustave je jednostavno vizualizirati i pratiti detekciju po čvorovima stabla ili pravilo po pravilo pa je lako zaključiti što je dovelo do detekcije [3]. Crne kutije predstavljaju sustave koje je teško ili nemoguće interpretirati, umjetne neuronske mreže spadaju u te sustave. Proces odlučivanja neuronske mreže može se približno opisati kroz vizualizaciju podataka i značajki koje sustav koristi za detekciju, te okvirno opisivanje veza između neurona [3]. Također je korisno uz sami rezultat klasifikacije, tj. detektirano stanje stroja, prikazati i vjerojatnost da je ta predikcija točna, ovime se opisuje koliko se „može vjerovati“ nekoj detekciji što je veoma korisno operateru pri odlučivanju.

Rad modela dijagnostike i detekcije potrebno je periodički provjeravati zato jer se performanse detekcije mogu mijenjati kroz vrijeme. Za ovu svrhu razvijaju se razni *monitoring* softveri. Performanse mogu vremenom padati zbog degradacije u hardveru (mijenjanje karakteristike senzora i kvar na senzoru), zato je potrebno pratiti ulazne vrijednosti podataka kako bi se izbjegle krive detekcije [3]. Također moguće je modele učiti *Online*, tj.za vrijeme rada stroja kako pristižu novi podatci [3].

3. OPIS PODATAKA U DATASETU

Dataset je preuzet sa UCI repozitorija za strojno učenje (*eng. UCI machine learning repository*) [4]. Podaci u datasetu generirani su pomoću „demonstratora“. Demonstrator je sklop koji se sastoji od elektromotora i njegove upravljačke jedinice, reduktora i tereta. Na demonstrator je moguće instalirati različite komponente, ispravne ili neispravne, i moguće ga je opteretiti različitim teretima. Ovaj sklop se koristi u edukacijske svrhe kako bi se mogli simulirati različiti uvjeti u industriji i razni kvarovi koji se mogu dogoditi na elektromotornom pogonu. Prikupljanje podataka se vrši tako što se mjere signali dviju ulaznih faznih struja na elektromotoru, pomoću osciloskopa i strujnih mjernih sondi. Dobivene signale je potrebno pred obraditi kako bi se iz njih izvukle značajke koje opisuju stanje stroja.

3.1. Predobrada podataka

Signali struje elektromotornog pogona su pomoću empirijske modalne dekompozicije rastavljeni na intrinzične frekvencijske komponente [5]. Signal se može rastavljati na frekvencijske komponente sve dok se ne izvuku sve značajke iz početnog signala, ali se testiranjem utvrdilo da je dovoljno koristiti prve tri frekvencijske komponente i ostatak od dekompozicije za adekvatan opis značajki signala. Frekvencijske komponente su dalje rastavljene na više manjih perioda, duljine jedne rotacije kugličnog ležaja na pogonu [5]. Za svaki period je izračunata srednja vrijednost, standardna devijacija, asimetričnost i spljoštenost. Dobivene vrijednosti su upisane u matricu značajki te razvrstane po stanju pogona kojem pripadaju. Pošto je dobiven velik broj značajki potrebno je nekako smanjiti broj značajki kako bi se smanjilo potrebno vrijeme za učenje neuronske mreže. Za određivanje koje kombinacije značajki najbolje razdvajaju definirane klase i smanjenje broja korištenih značajki koristi metoda analize linearne diskriminante [5]. Tako pripremljen dataset se nalazi u .txt datoteci na *Machine learning repository-u [4]*.

3.2. Statistička analiza podataka

Nad podacima iz dataseta provedena je statistička analiza, za svaki od podataka u setu izvučena je minimalna i maksimalna vrijednost, srednja vrijednost, standardna devijacija i varijanca. Statistička analiza provodi se kako bi se podaci mogli bolje predočiti. Rezultat statističke analize vidljiv je u DODATKU A – Tablica statističke analize.

4. ANALIZA GLAVNIH KOMPONENTI (PCA)

Analiza glavnih komponenti *(eng. Principal Component Analysis)* je metoda smanjivanja dimenzija nekog dataseta, tj. smanjivanja broja varijabli u nekom datasetu pretvaranjem početnih varijabli u druge varijable kojih ima manje ali sadržavaju većinu informacija koje sadržava početni dataset [6]. Razlog smanjivanja varijabli je pojednostavljenje dataseta i ubrzavanje treniranja nekog algoritma, no ako se dataset previše smanji gubi se previše podataka iz originalnog dataseta i može doći do manje točnosti algoritma.

U načelu se analiza glavnih komponenti može podijeliti u 5 koraka:

1. Standardizacija ulaznih varijabli
2. Računanje matrice kovarijanci
3. Računanje svojstvenih vektora i svojstvenih vrijednosti matrice kovarijanci
4. Određivanje vektora značajki za odabir glavnih komponenti
5. Prerada podataka pomoću glavnih komponenti

4.1. Standardizacija ulaznih varijabli

Standardizacija je potrebna zbog toga što je analiza glavnih komponenti jako osjetljiva na nestandardizirane podatke, tj. na veliku razliku između opsega vrijednosti ulaznih podataka. Standardizaciju je najlakše postići oduzimanjem srednje vrijednosti od svake varijable i dijeljenjem s standardnom devijacijom.

4.2. Računanje matrice kovarijanci

Računanje matrice kovarijanci se provodi kako bi se odredilo koliko ulazne varijable ovise jedne o drugima, tj. koliko su korelirane. Ponekad su varijable korelirane na takav način da sadrže redundantne informacije te se to može iskoristiti za smanjenje dimenzija dataseta. Matrica kovarijanci je n×n simetrična matrica (n je broj dimenzija dataseta) koja sadrži kovarijance svih mogućih kombinacija varijabli u datasetu. Na glavnoj dijagonali matrice nalaze se varijance svake varijable, a matrica je simetrična oko glavne dijagonale.

4.3. Računanje svojstvenih vektora i svojstvenih vrijednosti matrice kovarijanci

Glavne komponente su linearna kombinacija ulaznih varijabli koje su međusobno ne korelirane. Analiza glavnih komponenti nastoji u prvoj glavnoj komponenti pohraniti najviše moguće informacija, zatim u sljedećoj komponenti najviše moguće ostatka informacija i tako sve do originalnog broja podataka. Originalni dataset dimenzije n ima n glavnih komponenti, ali nije potrebno koristiti sve glavne komponente u analizi pošto je većina informacija iz dataseta zadržana među prvim glavnim komponentama. Svojstveni vektori i svojstvene vrijednosti opisuju glavne komponente, što je svojstvena vrijednost veća to je više podataka sadržano u glavnoj komponenti, a svojstveni vektor opisuje smjer osi u prostoru koja sadrži varijancu podataka opisanu u svojstvenoj vrijednosti. Raspoređivanjem osnovnih komponenti u odnosu na svojstvene vrijednosti, od veće prema manjoj dobivaju se osnovne komponente raspoređene po značajnosti, tj. po količini sadržanih podataka. Određivanje postotne varijance koju neka komponenta sadrži moguće je napraviti na način da se svojstvena vrijednost podijeli sa sumom svih svojstvenih vrijednosti. Pomoću postotne varijance je moguće odrediti koliko glavnih komponenti će se koristiti u analizi i koliko će originalne varijance zadržati konačni prerađeni dataset.

4.4. Određivanje vektora značajki za odabir glavnih komponenti

Vektor značajki se sastoji od svojstvenih vektora u stupcima te ima broj stupaca jednak broju komponenti koje se zadržavaju u analizi. Vektorom značajki se određuju konačne dimenzije podataka i postotak zadržane varijance u konačnom prerađenom datasetu.

4.5. Prerada podataka pomoću glavnih komponenti

U ovom koraku se prvi put mijenjaju ulazni podaci nakon standardizacije. Prerada podataka se izvršava na način da se transponirana matrica vektora značajki množi s transponiranom matricom originalnog standardiziranog dataseta:

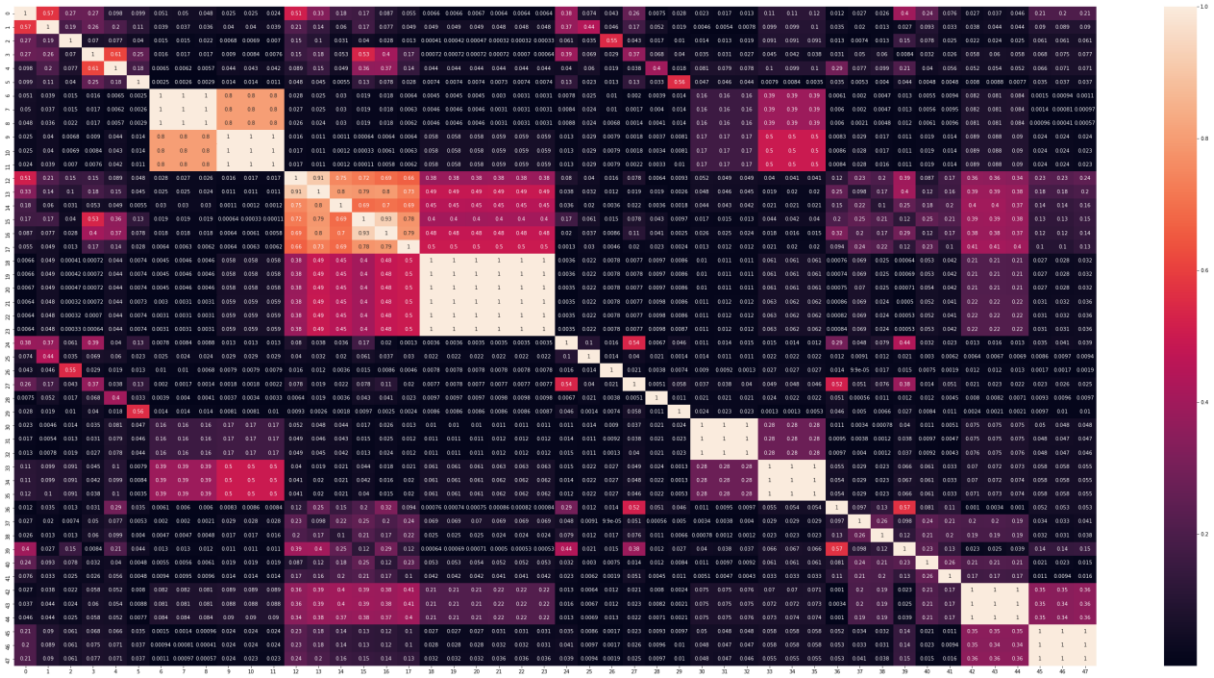
$$[Prerađeni\ dataset] = [Vektor\ značajki]^T * [Standardizirani\ originalni\ dataset]^T. \quad (4.1)$$

Primjer korištenog koda za preradu podataka pomoću glavnih komponenti [6]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=0)
# skaliranje podataka
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# pca

pca = PCA(.97)

pca.fit(X_train)

X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

5. UKLANJANJE JAKO KORELIRANIH PODATAKA

Ukoliko se u datasetu nalaze značajke koje jako linearno ovise jedna o drugoj to može smanjiti brzinu treniranja nekog algoritma i smanjiti postignutu točnost nekog algoritma. Povezanost podataka se u statistici naziva korelacija, a koeficijent korelacije predstavlja stupanj povezanosti nekih podataka [7]. Koeficijent korelacije ima raspon vrijednosti od 0 do 1, gdje 1 predstavlja potpuno korelirane podatke a 0 potpuno nekorelirane podatke. Kako bi se korelacija podataka mogla vizualizirati korisno je prikazati matricu korelacije, koja prikazuje koliko pojedini podatak u datasetu ovisi o bilo kojem drugom podatku u datasetu [7]. Korelacijska matrica je prikazana na slici 5.1., pojedine ćelije obojene su različitim bojama kako bi se lakše razaznalo koliki je koeficijent korelacije za svaki podatak u datasetu.



Slika 5.1. Korelacijska matrica

Uvećana korelacijska matrica nalazi se u DODATKU B. Korelacijska matrica je zrcalna oko dijagonale, i elementi na samoj dijagonali iznose 1. Kako je matrica zrcalna dovoljno je promatrati samo gornju ili donju trokutastu matricu bez elemenata na dijagonali kako bi utvrdili koji podaci su korelirani. Sve ćelije bijele ili svijetlo narančaste boje predstavljaju jako korelirane podatke, stoga je iz korelacijske matrice vidljivo da u datasetu ima dosta koreliranih podataka koji usporavaju algoritam te bi ih bilo korisno ukloniti. Uklanjanje se vrši jednostavno tako da se uklone stupci iz dataseta koji u korelacijskoj matrici imaju koeficijent korelacije veći od 0.95 [8].

Primjer korištenog koda za uklanjanje koreliranih podataka [8] :

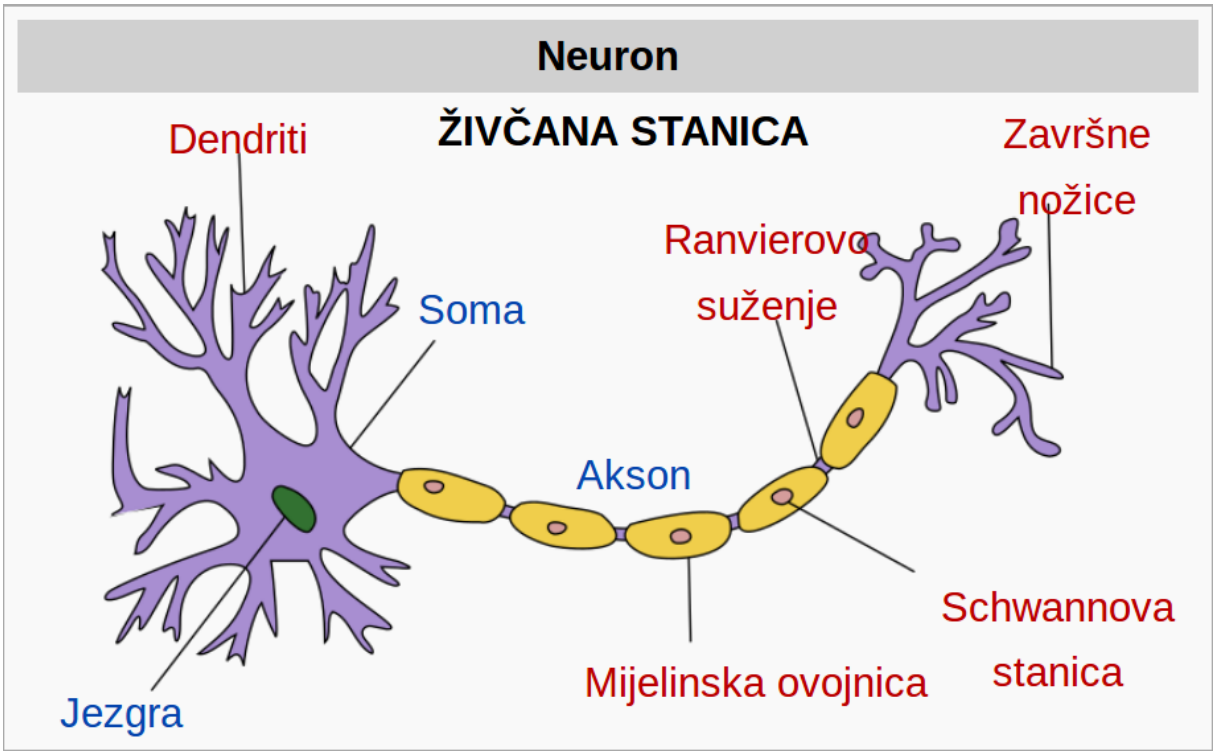
```
df= pd.DataFrame(X_org)
print(); print(df.head())
corrMatrix = df.corr().abs()
plt.figure(figsize=(48,24))
sn.heatmap(corrMatrix, annot=True)
plt.show()

upper_tri =
corrMatrix.where(np.triu(np.ones(corrMatrix.shape),k=1).astype(np.bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] >
0.95)]
X = df.drop(df.columns[to_drop], axis=1)
```

6. NEURONSKE MREŽE I VIŠESLOJNI PERCEPTRON

Umjetne neuronske mreže nastale su iz želje da se imitira ljudski mozak pri izvođenju kompliciranih operacija poput odlučivanja i učenja. Razvoj umjetnih neuronskih mreža počeo je od 1940. godine na dalje [2]. Umjetne neuronske mreže su u suštini algoritmi koji su dizajnirani da prepoznaju skrivene uzorke u podacima [9]. Podaci koje umjetne neuronske mreže interpretiraju su numeričkog tipa, sadržani u vektorima značajki, što znači da se podaci iz pravog svijeta poput slika, zvuka ili signala u vremenu moraju pretvoriti u numeričke podatke kako bi ih umjetne neuronske mreže mogle analizirati. Pomoću neuronskih mreža podaci se mogu razvrstati u skupine sličnih podataka, ukoliko postoji informacija o nazivima tih skupova podataka onda algoritam može klasificirati nove podatke. Umjetne neuronske mreže se također mogu koristiti i za regresiju, tj. predviđanje kretanja nekog podatka. Neuronsko računarstvo se ne smatra kao konkurencija klasičnim računalnim metodama, već kao dopuna postojeće tehnologije [2].

6.1. Biološki neuron

Biološki neuron je temeljna stanica u ljudskom mozgu. Skica biološkog neurona prikazana je na slici 6.1.

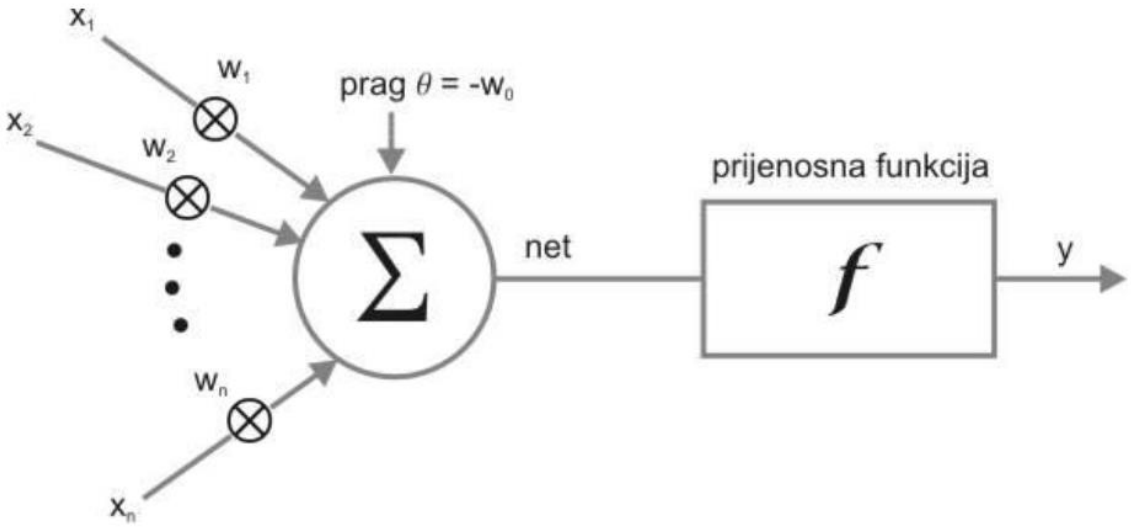


Slika 6.1. Biološki neuron [10]

Ljudski mozak sačinjen je od deset miliona međusobno povezanih neurona, neuron predstavlja stanicu koja prima informacije, procesira ih i šalje informacije. Dendriti predstavljaju ulaze u neuron, primaju informacije s drugih neurona. Stanično tijelo obrađuje informacije sa ulaza i generira izlaznu informaciju koju preko aksona šalje na izlazne sinapse. Sinapse su mali prostori između neurona kojima se prenose informacije pomoću neurotransmitera.

6.2. Umjetne neuronske mreže

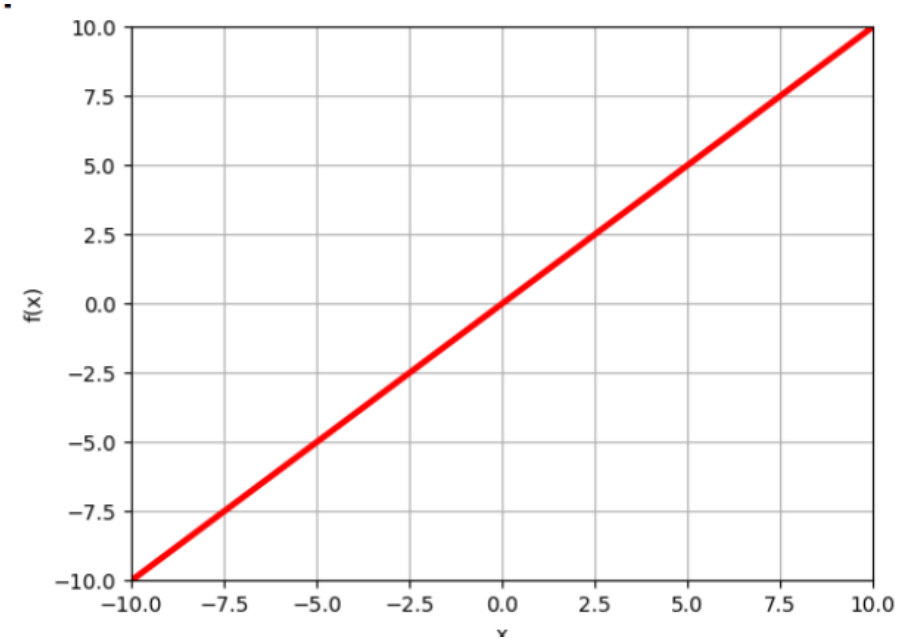
Umjetne neuronske mreže sastoje se od jednog ili više neurona koji su međusobno povezani, neuroni se postavljaju u slojeve [2]. Neuroni predstavljaju jednostavne jedinice za obradu informacija. Blok dijagram neurona prikazan je na slici 6.2.



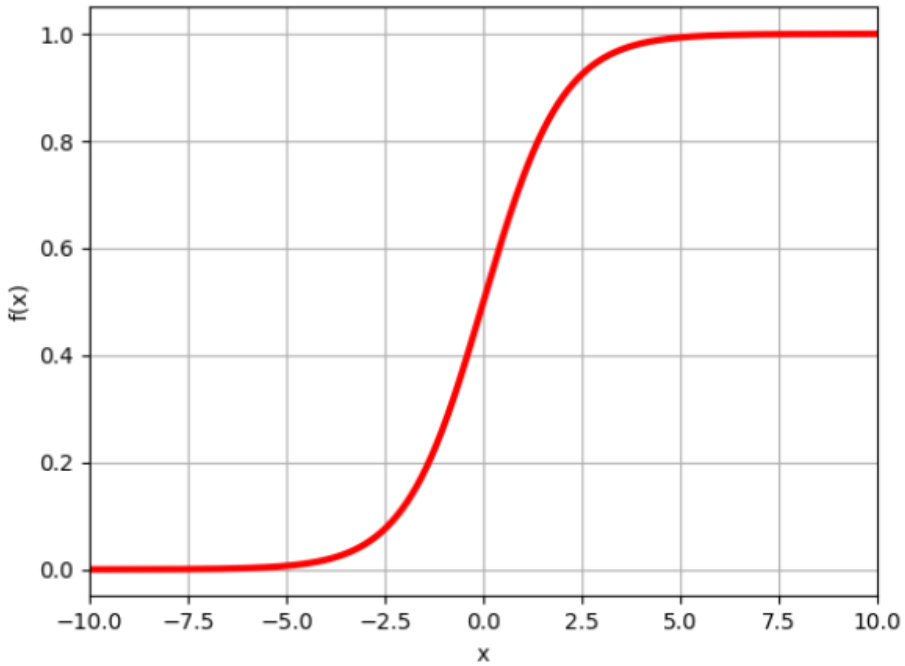
Slika 6.2. Umjetni neuron [2]

Ulazne vrijednosti su označene s X, njihovi težinski faktori s W a aktivacijska funkcija označena je s f. Svaki ulaz u neuron ima težinski faktor koji regulira jačinu svake ulazne vrijednosti, tako modificirane ulazne vrijednosti se sumiraju u neuronu i dodaje im se „*Offset*“ tj. odstupanje. Rezultat od sumacije se dovodi na aktivacijsku funkciju koja generira izlazni signal. Kod treniranja umjetne neuronske mreže prvo se svi težinski faktori i odstupanja od svih neurona nasumično biraju, a zatim se postupno modificiraju u cilju da nova iteracija postigne bolji rezultat od prethodne. Kad su težinski faktori i odstupanja podešeni na način da se postigao najbolji mogući rezultat, znači da se postiglo stanje da je određeni niz naučen. Aktivacijska funkcija predstavlja matematička „vrata“ između ulaza koji napaja trenutni

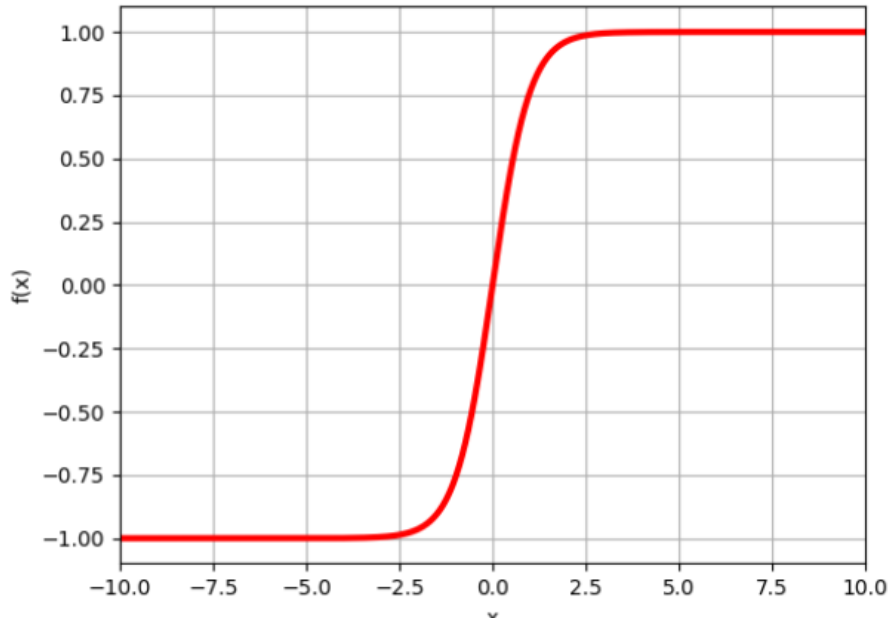
neuron i njegovog izlaza koji prelazi u sljedeći sloj [2]. Aktivacijske funkcije su najčešće nelinearne ali postoje i linearne. Linearne aktivacijske funkcije su najčešće ograničene zbog proporcionalnosti izlaznog i ulaznog signala. Neke od aktivacijskih funkcija prikazane su na slikama 6.3., 6.4., 6.5., 6.6.



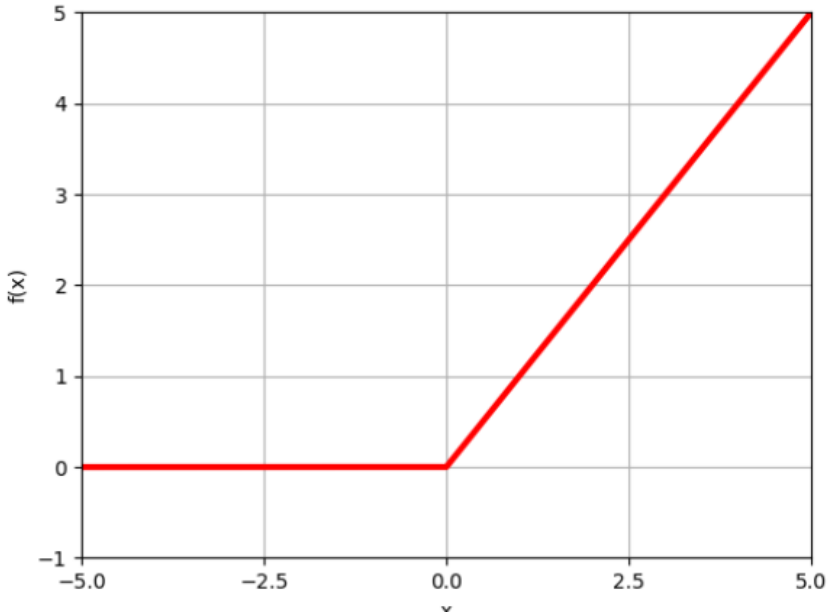
Slika 6.3. Linearna aktivacijska funkcija [2]



Slika 6.4. Sigmoidalna aktivacijska funkcija [2]



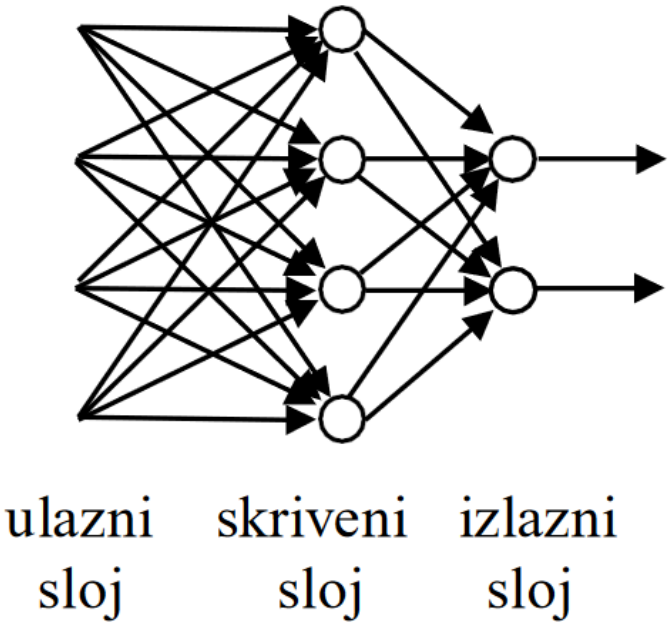
Slika 6.5. Tanh aktivacijska funkcija [2]



Slika 6.6. ReLU aktivacijska funkcija [2]

6.3. Višeslojne neuronske mreže (*Multi Layer Perceptron*)

Višeslojne neuronske mreže imaju uz ulazni i izlazni sloj i skrivene slojeve. Izlazi iz jednog sloja predstavljaju ulaze u drugi sloj [2]. Neuronska mreža s jednim skrivenim slojem prikazana je na slici 6.7.

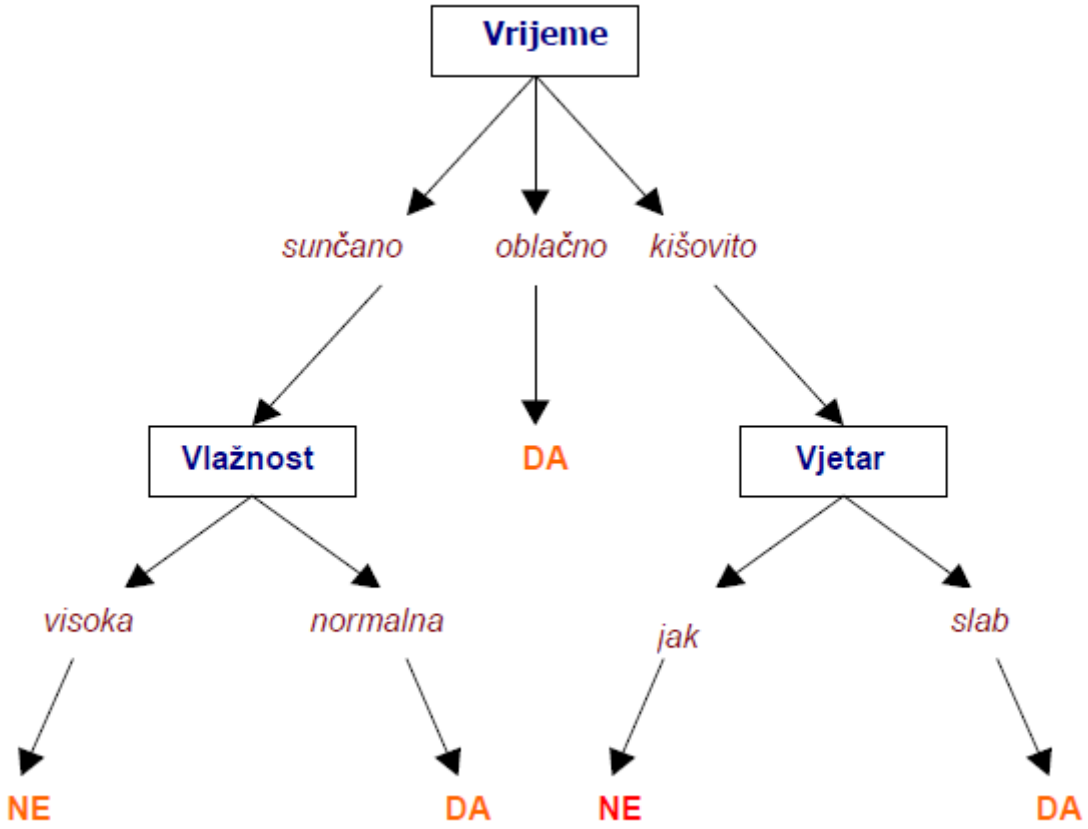


Slika 6.7. Mreža s jednim skrivenim slojem [2]

Prvi sloj je ulazni sloj u kojem broj neurona odgovara broju ulaza u mrežu. Svaki izlaz iz neurona u nekom sloju povezan je na ulaz svakog neurona u idućem sloju. Neuroni u skrivenim slojevima transformiraju sume ulaznih vrijednosti i težinskih faktora primjenom aktivacijske funkcije [2]. Izlazni sloj prima vrijednosti od skrivenog sloja i transformira ih u izlazne vrijednosti [2]. Višeslojne neuronske mreže su algoritmi koji se uče s učiteljem. Prednosti višeslojnih neuronskih mreža su sposobnost da se istrenira za nelinearne modele i sposobnost da se istrenira u realnom vremenu koristeći opciju „*partial-fit*“ [2]. Nedostatak je što je potrebno optimizirati hiperparametre kako bi se dobilo što bolje rješenje, što može oduzeti dosta vremena prilikom dizajna modela.

7. STABLA ODLUKE (*DECISION TREES*)

Stabla odluke su najraširenija metoda induktivnog zaključivanja, najčešće primjene ima u medicini ili financijama [11]. Stabla odluke se sastoje od čvorova (*eng. node*) i grana (*eng. branch*). Čvorovi predstavljaju atribut koji se testira, a grane vrijednost testiranog atributa. Klasifikacija nekog problema obavlja se odozgo prema dole, tj. od korijena prema listovima. Korijen je početni atribut a listovi su rezultati klasifikacije. Primjer stabla odluke vidljiv je na slici 7.1.



Slika 7.1. Primjer stabla odluke [11]

Primjer prikazan na slici 7.1. prikazuje problem klasifikacije „Da li je jutro pogodno za tenis?“ [11]. Atribut Vrijeme predstavlja korijen stabla a odgovori DA i NE rezultat klasifikacije, tj. listove. Podaci za učenje su predstavljeni parovima atribut-vrijednost.

7.1. Učenje stabla odluke

Temeljni algoritam učenja stabla odluke predstavio je Quinlan 1986. u [12] te ga je nazvao ID3(*eng. Induction of Decision Trees*). Atributi se odabiru na način da se testira svaki atribut da se ocijeni kako dobro klasificira primjere te se najbolji odabire kao čvor a njegove vrijednosti su silazne grane po kojima se sortiraju primjeri za daljnje učenje te se postupak dalje ponavlja za te primjere [11]. ID3 spada u pohlepne(*eng. greedy*) algoritme jer se nikada ne vraća da bi ponovno razmotrio prethodne čvorove.

Atributi koji se testiraju u pojedinom čvoru stabla se najčešće odabiru na temelju informacijske dobiti. Informacijska dobit je mjera koliko dobro pojedini atribut odjeljuje primjere za učenje u skladu s ciljnom klasifikacijom [11], može se još predstaviti kao očekivana redukcija entropije uzrokovana podjelom primjera za učenje. Entropija je mjera „neurednosti“ podataka, tj. u ovom slučaju mjera homogenosti primjera za učenje stabla odluke [11]. Matematički opis entropije dan je izrazom [11]:

$$Entropija(S) = \sum_{i=1}^{c} -p_i \log_2 p_i. \quad (7.1)$$

Gdje je:

S – skup primjera nekog ciljnog koncepta,

c – broj klasa primjera u skupu S,

p_i – proporcija primjera koji pripadaju i-toj klasi.

Informacijska dobit atributa A u odnosu na skup primjera S dana je izrazom [11]:

$$Informacijska_{dobit(S,A)} = Entropija(S) - \sum_{v \in Vrijednost(A)} \frac{|S_v|}{|S|} Entropija(S_v). \quad (7.2)$$

Gdje je:

Vrijednost(A) – skup svih mogućih vrijednosti atributa A,

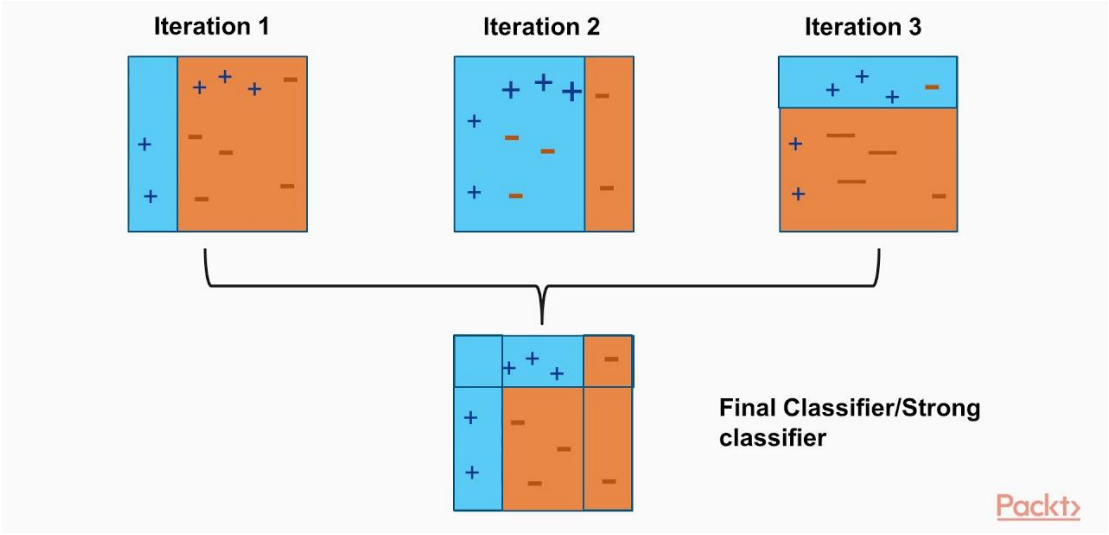
S_v – podskup od S za koji atribut A ima vrijednost v.

Promatrani atribut je bolji što je veća vrijednost informacijske dobiti, te se za neki čvor odabire atribut s najvećom informacijskom dobiti.

Mogući problem prilikom učenja stabla odluke predstavlja prenaučenost. Kod ID3 algoritma stablo raste dok se svi podaci pravilno ne klasificiraju, što može biti problem kod podataka s greškom ili premalog broja podataka. Definicija prenaučenosti: Neka je dan prostor hipoteza H, hipoteza h Є H je prenaučena ako postoji hipoteza h' Є H takva da h ima manju pogrešku nego h' na primjerima za učenje, ali h' ima manju pogrešku nego h na cijelom prostoru primjera [11]. Postoje dva pristupa za izbjegavanje prenaučenosti, zaustavljanje rasta stabla prije savršene klasifikacije primjera za učenje (*eng. Early stopping*) i naknadno podrezivanje prenaučenog stabla. Naknadno podrezivanje pokazalo se učinkovitijom metodom. Metoda naknadnog podrezivanja temelji se na podjeli skupa dostupnih primjera na tri skupa podataka, skup za učenje, skup za validaciju i skup za testiranje. Stablo se uči svim podacima iz skupa za učenje, zatim se validira podacima iz skupa za validaciju te se podrezuje u ovisnosti o točnosti klasifikacije. Stablo se podrezuje sve dok točnost klasifikacije ne počne smanjivati, zatim se tako podrezano stablo testira na podacima iz skupa za testiranje.

7.2. Podizanje gradijenta (Gradient Boosting)

Ideja kod podizanja gradijenta je da se pomoću više loših klasifikatora može postići dobra klasifikacija nekog problema. Loš klasifikator je definiran kao klasifikator čije su performanse barem malo bolje od potpuno slučajne klasifikacije. Prvi algoritam koji se bavio podizanjem (*eng. Boosting*) bio je AdaBoost. Loši klasifikatori koji se koriste u AdaBoost algoritmu su stabla odluke sa samo jednim čvorom, mogu se još nazivati i panjevi odluke (*eng. Descision Stumps*) [13]. Na slici 7.2. prikazan je dijagram koji objašnjava rad AdaBoost algoritma.



Slika 7.2. AdaBoost algoritam[14]

U prvoj iteraciji na slici (*eng. Iteration 1*), koja predstavlja prvu predikciju panja odluke, vidi se da je panj točno klasificirao dva pozitivna podatka i pet negativnih podataka ali je krivo klasificirao tri pozitivna podatka kao negativne. Na tri krivo klasificirana pozitivna podatka povećavaju se težinski faktori te se primjenjuje novi panj odluke čiji je rezultat vidljiv u iteraciji 2 (*eng. Iteration 2*). Na slici su veći težinski faktori prikazani s većim simbolom za podatke. Ovaj puta su točno klasificirani podaci s većim težinskim faktorima ali su krivo klasificirana tri negativna podatka kao pozitivni, tako da se na tim negativnim podacima opet povećavaju težinski faktori te se opet primjenjuje novi panj odluke. Postupak se ponavlja na isti način dok god se ne postigne potpuno točna klasifikacija, koja je vidljiva u završnom klasifikatoru (*eng. Final Classifier*) [15].

Podizanje gradijenta (*eng. Gradient boosting*) radi na sličan način kao i AdaBoost algoritam, samo za odabir težinskih faktora koristi funkciju gubitka (*eng. Loss function*) i gradijentni spust. Gradijent preuzima ulogu derivacije ukoliko funkcija sadrži više varijabli, gdje isti određuje nagib same tangente grafa funkcije i prikazuje smjer najvećeg povećanja funkcije [2]. Gradijentni spust optimizacijski je algoritam kojem je glavna svrha pronalaženje minimuma derivabilne funkcije tj. globalnog optimuma [2]. Kod algoritma podizanja gradijenta modeli strojnog učenja se treniraju sekvencijalno te se na svakoj iteraciji postupno smanjuje funkcija gubitaka koristeći metodu gradijentnog spusta [13]. Kao model strojnog učenja koriste se stabla odluke, ali veća nego kao kod AdaBoosta – do 8 razina čvorova. Algoritam i dalje spada u pohlepne algoritme i stabla se najčešće ograničavaju na neki način, poput maksimalnog broja slojeva, razina ili listova. Stabla se ograničavaju kako bi se

osiguralo da modeli ostanu neprecizni [13]. Stabla se dodaju jedno po jedno, te se pritom ne mijenjaju prethodno stvorena stabla. Izlaz iz novog stabla se pridodaje na izlaze prethodnih stabala u cilju da se ispravi ili poboljša finalni izlaz iz modela [13]. Treniranje završava kada se dosegne neki predefinirani broj stabala ili kada se dostigne zadovoljavajuća razina dobitka ili kada se ne može više unaprijediti model daljnjim učenjem.

7.3. XGBOOST

XGBOOST (*eng. eXtreme Gradient Boosting*) je algoritam temeljen na stablima odluke i podizanju gradijenta. Algoritam podržava tri oblika podizanja, podizanje gradijenta, stohastičko podizanje gradijenta i regularizirano podizanje gradijenta. Glavne razlike u odnosu na samo podizanje gradijenta su velika brzina učenja i velika točnost modela.

8. GRIDSEARCH

GridsearchCV je algoritam za pretraživanje prostora hiperparametara kako bi se pronašli hiperparametri koji daju najbolju klasifikaciju za neki određeni slučaj. Gridsearch se može koristiti s bilo kojim algoritmom strojnog učenja, samo je potrebno adekvatno zadati prostor hiperparametara koji se pretražuje i sami model koji će se koristiti.

Primjer korištenog koda za pretraživanje hiperparametara pomoću gridsearch algoritma (za mlp algoritam) :

```
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'hidden_layer_sizes':  [(40,40,40,40,40), (40,40,40,40),
                                        (20,20,20,20,20), (20,20,20,20),
                                        (20,40,100,40,20), (20,40,100),
(20,40,20), (40,100,20)],
               'activation': ['relu','identity','logistic','tanh'],
               'solver': ['adam', 'lbfgs'],
               'learning_rate':['constant','adaptive','invscaling'],
               'learning_rate_init': [0.1,0.01,0.5, 0.00001],
               'alpha': [0.001,0.002,0.00205, 0.003],
               'max_iter': [500]}]


model = GridSearchCV(MLPClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
```

Osim hiperparametara potrebno je zadati na koliko dijelova će se razdvojiti dataset za treniranje i testiranje, i koji će se kriterij koristiti za ocjenjivanje pojedinog modela. U gornjem slučaju odabrano je dijeljenje dataseta na 10 jednakih dijelova (uz miješanje podataka) i kriterij ocjenjivanja *accuracy* tj. preciznost klasifikacije.

9. PRIMJENA ALGORITAMA

Prije primjene algoritama potrebno je učitati dataset u python radno okruženje.

Primjer korištenog koda za učitavanje dataseta iz txt datoteke:

```
Dataset = open("Sensorless_drive_diagnosis.txt")

X = []
Y = []

for ln in dataset.readlines():
    ct = 0
    x = []
    for dt in ln.split():
        ct = ct+1
        if ct == 49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x = np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X = np.array(X)
Y = np.array(Y)
```

Na dataset su primjenjeni algoritmi za pred obradu podataka i algoritmi strojnog učenja opisani u prethodnim poglavljima. Algoritmi strojnog učenja koji su korišteni su *MLPClassifier* i *XGBClassifier*. Oba algoritma su primijenjena na originalni set podataka, set obrađen analizom glavnih komponenti i set s uklonjenim koreliranim podacima.

Primjer korištenog koda za treniranje *MLPClassifier* algoritma:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

#Unos parametara klasifikatora
clf = MLPClassifier(random_state=0, alpha=0.00205, solver='adam',
                    hidden_layer_sizes=(20,40,100,40,20),
                    activation='tanh', verbose='True',
                    learning_rate='constant', max_iter=500)

# Treniranje klasifikatora s podacima za treniranje
clf.fit(X_train, y_train)
```

Primjer korištenog koda za treniranje *XGBClassifier* algoritma:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

#Unos parametara klasifikatora
clf1= XGBClassifier( colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, learning_rate=0.05,
              max_depth=10, min_child_weight=8,
              n_estimators=100,
              random_state=0, reg_alpha=0, reg_lambda=1,
              subsample=0.5,
              verbosity=0)


#Treniranje klasifikatora s podacima za treniranje
clf1.fit(X_train,y_train)
```

Optimizacija hiperparametara je prvo napravljena ručno a zatim su se odabrani parametri dodatno provjeravali pomoću *GridsearchCV* algoritma koji je izvršavan na superračunalu „Bura“. Korišteni hiperparametri za MLPClassifier i svaki dataset dani su u tablici 9.1.

Tablica 9.1. Korišteni hiperparametri za MLPClassifier algoritam

	Skriveni slojevi	Aktivacija	Solver	Learning rate, init	Alpha
Potpuni Dataset	40, 40, 40, 40	logistic	adam	Constant, 0.01	0.0001
PCA	20, 40, 100, 40, 20	tanh	lbfgs	Adaptive, 0.00001	0.0205
Korelacija	40, 100, 20	logistic	adam	Invscaling, 0.01	0.0001

Korišteni hiperparametri za *XGBClassifier* algoritam dani su u tablici 9.2.

Tablica 9.2. Korišteni hiperparametri za XGBClassifier algoritam

	Potpuni dataset	PCA	Korelacija
Booster	'gbtree'	'gbtree'	'gbtree'
Learning rate	0.05	0.05	0.05
Max depth	10	10	10
Min child weight	8	8	8
Colsample by level	1	1	1
Colsample by node	1	1	1
Colsample by tree	1	1	1
Reg alpha	0	0	0
Reg lambda	1	1	1
Broj estimatora	600	600	600

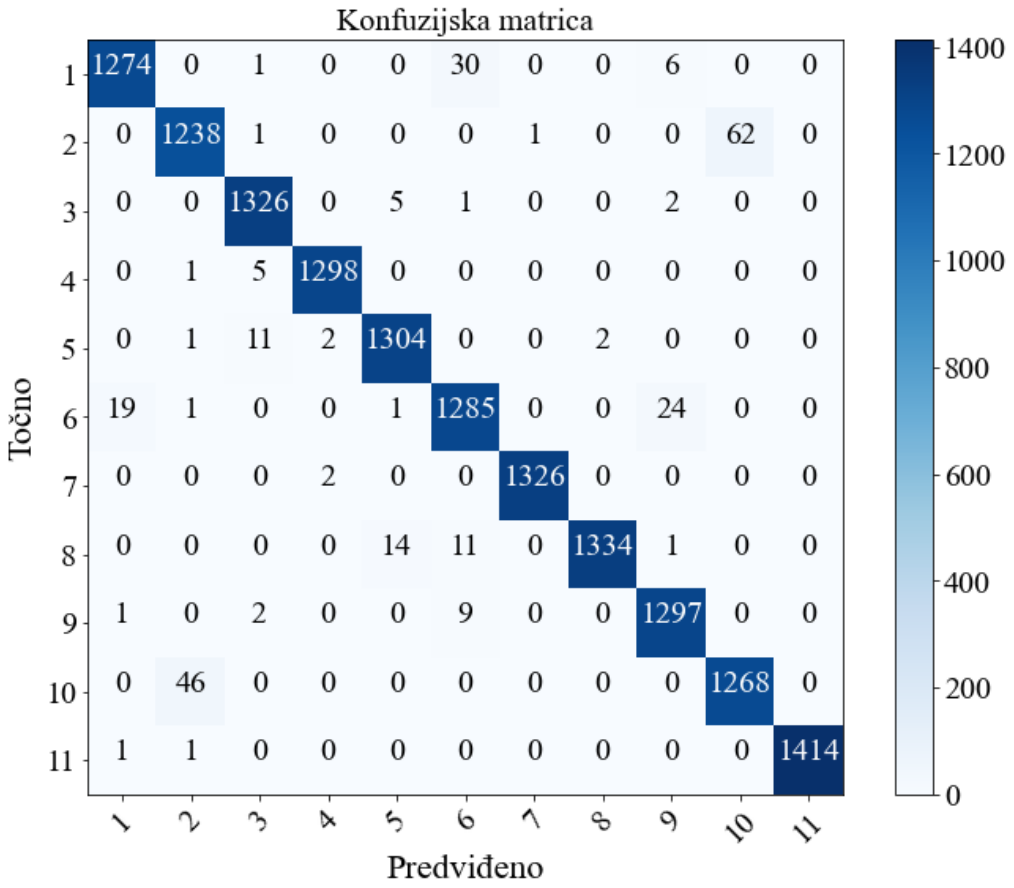
Hiperparametri za *XGBClassifier* algoritam odabrani su ručno i jednaki su za sva tri dataseta. Kod ovog algoritma nisu prikazani parametri odabrani pomoću *GridsearchCV* algoritma zbog tehničkih poteškoća pri izvođenju ovog klasifikatora na superračunalu, tj. zbog pre dugog vremena izvođenja *GridsearchCV* algoritma.

Algoritmi sa opisanim parametrima testirani su na podacima za test i dali su točnosti koje su prikazane u tablici 9.3.

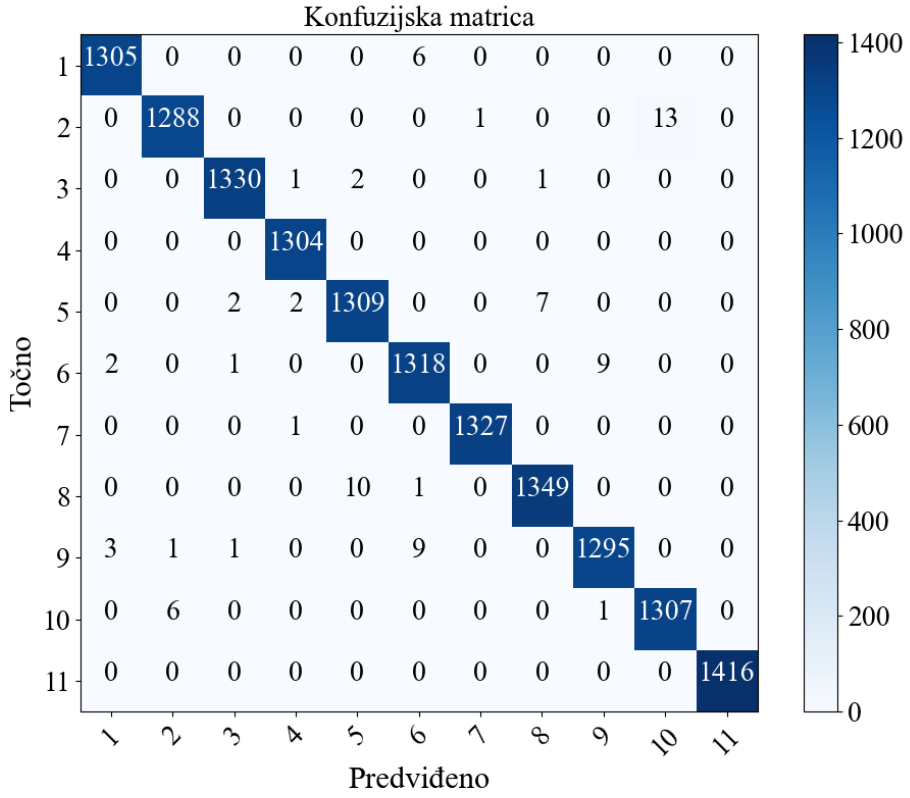
Tablica 9.3. Rezultati testiranja

	MLPClassifier	XGBClassifier
Potpuni dataset	98,20%, 123 iteracije	99,79 %
PCA	99,45%, 500 iteracija	97,03 %
Korelacija	98,09% 144 iteracija	99,82 %

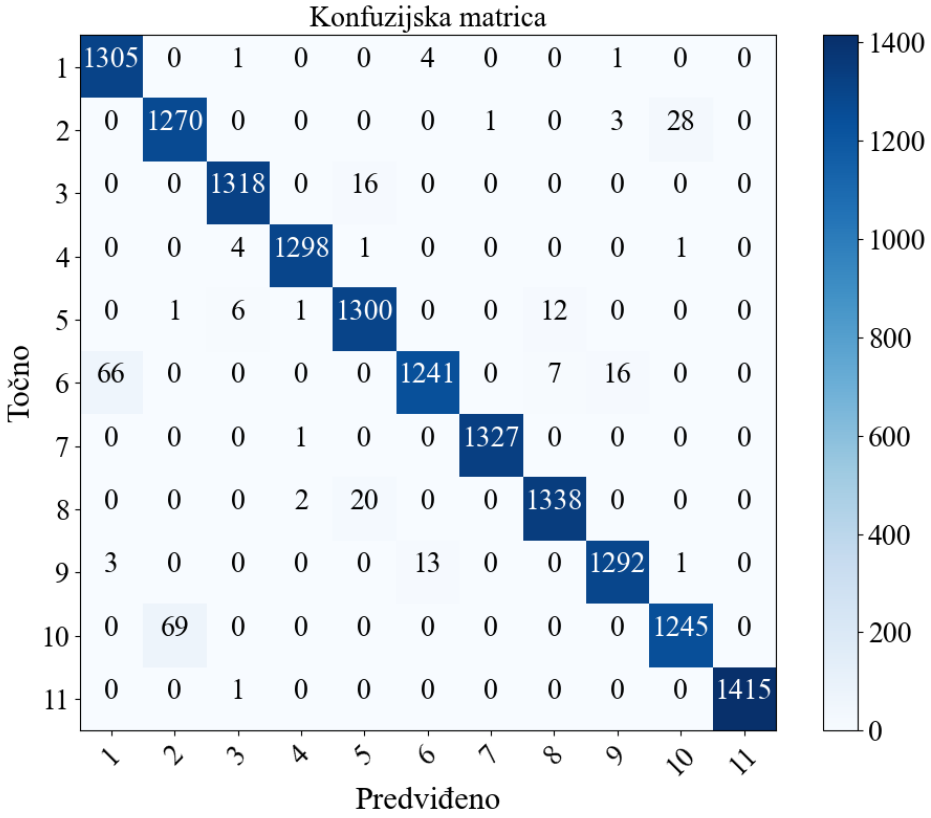
Na slikama 9.1, 9.2, i 9.3 prikazane su konfuzijske matrice za klasifikaciju MLPClassifier algoritmom za ukupni dataset, PCA dataset i korelacijski dataset.



Slika 9.1. Konfuzijska matrica za MLP algoritam i potpuni dataset



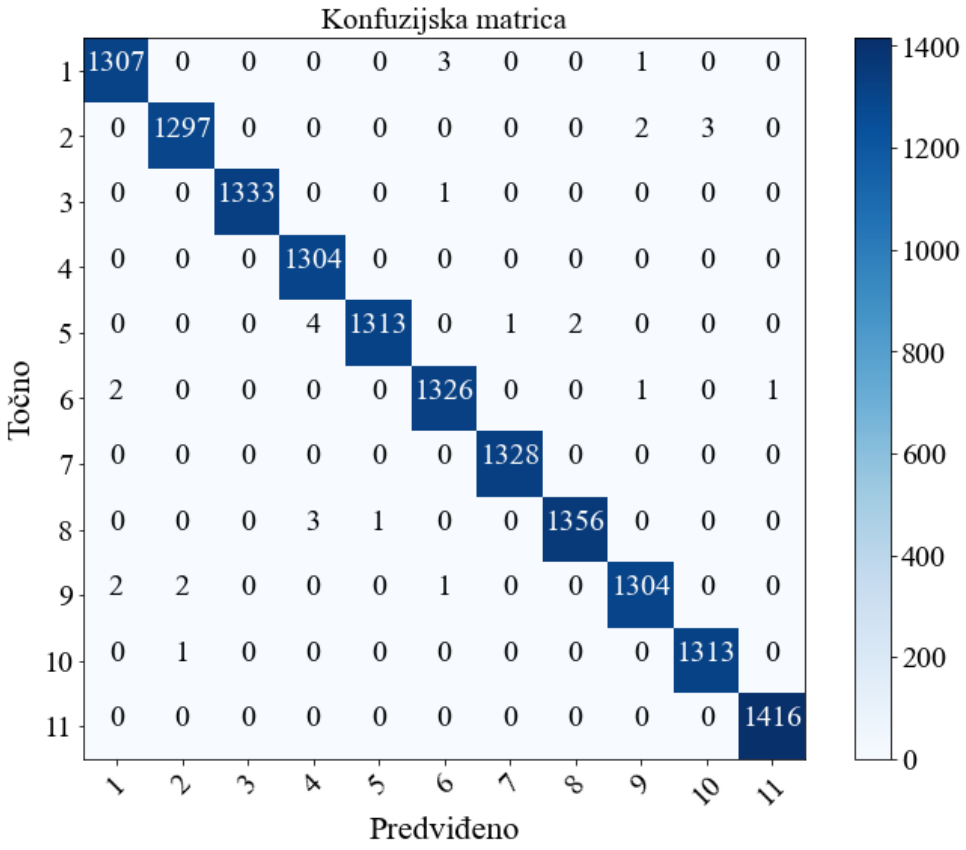
Slika 9.2. Konfuzijska matrica za MLP algoritam i PCA dataset



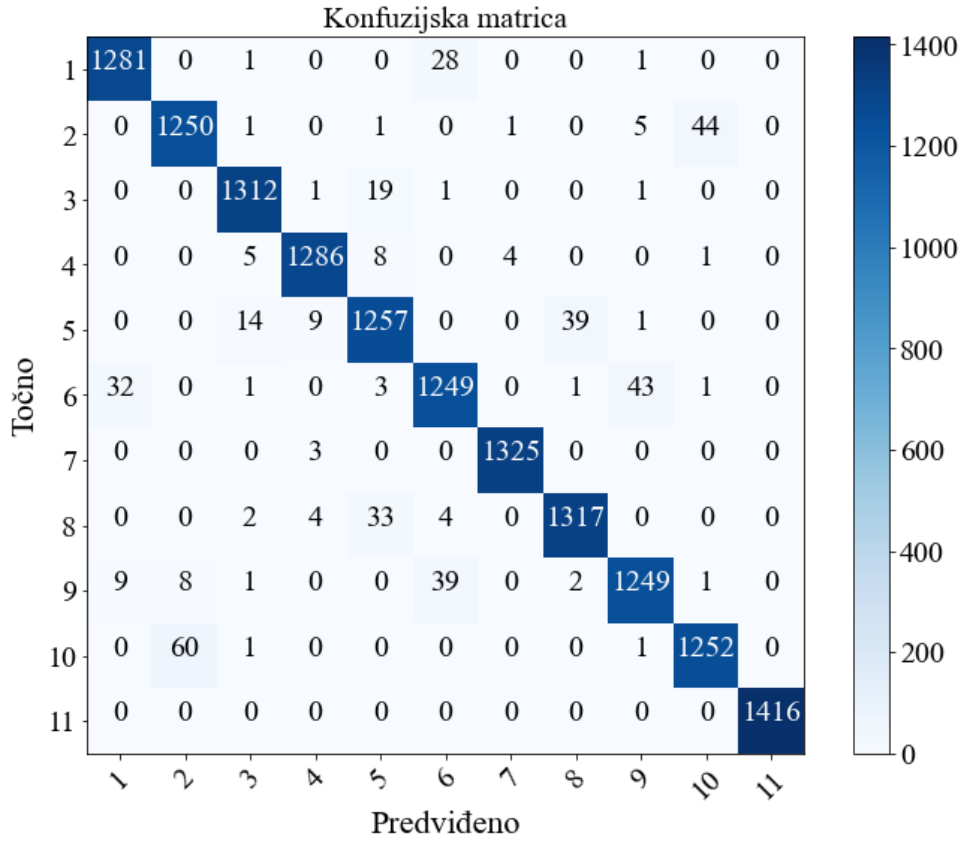
Slika 9.3. Konfuzijska matrica za MLP algoritam i Korelacijski dataset

Iz rezultata je vidljivo da MLP algoritam s PCA pred obradom daje najbolju klasifikaciju zadanog dataseta, dok je algoritam sa potpunim datasetom postigao najveću moguću točnost u najmanje iteracija učenja. MLP algoritam s PCA predobrađenim podatcima dosegao je maksimalni dozvoljeni broj iteracija i najsporije je završio proces učenja od svih ostalih slučajeva, ovo se dogodilo zbog toga što je kao *solver* odabran „*lbfgs*“ koji je općenito sporiji od drugih *solvera*, ali je u ovom slučaju dao najbolji mogući rezultat klasifikacije pa ga je iz tog razloga algoritam *GridSearchCV* odabrao. Iz konfuzijskih matrica za sva tri slučaja vidljivo je da je algoritam točno predvidio većinu podataka u datasetu za testiranje, dok je algoritam korišten s PCA predobrađenim podatcima imao najmanje netočno klasificiranih podataka.

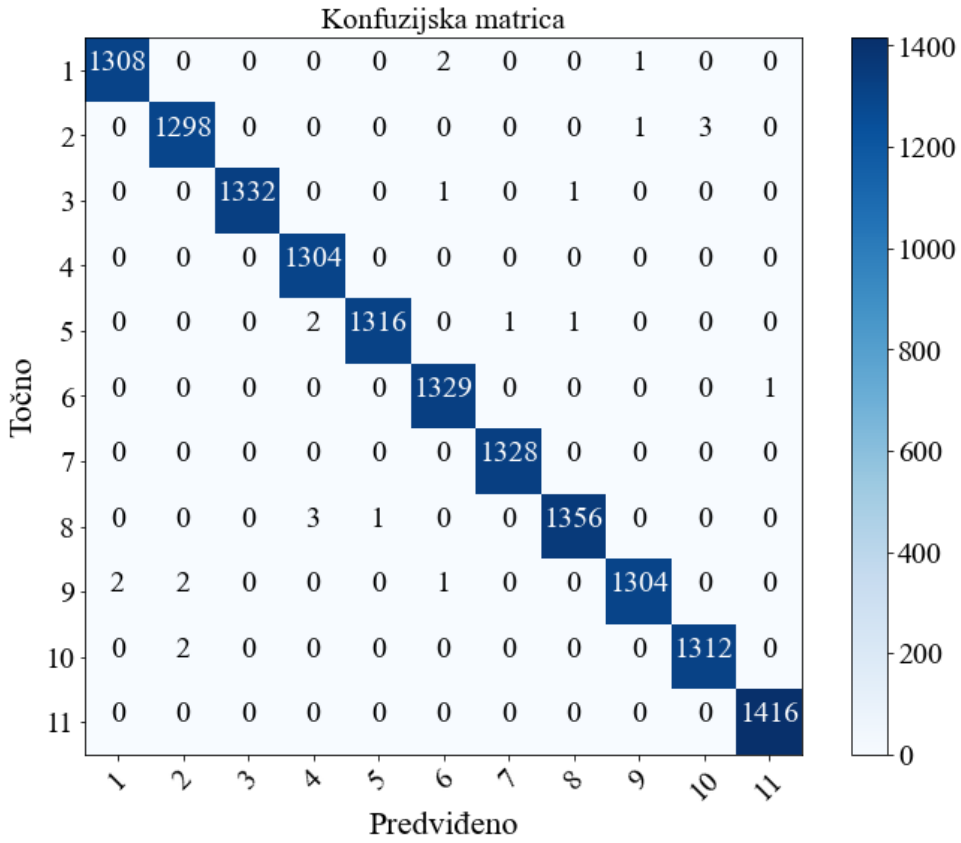
Na slikama 9.4, 9.5, i 9.6 prikazane su konfuzijske matrice za klasifikaciju XGBClassifier algoritmom za ukupni dataset, PCA dataset i korelacijski dataset.



Slika 9.4. Konfuzijska matrica za XGBoost algoritam i potpuni dataset



Slika 9.5. Konfuzijska matrica za XGBoost algoritam i PCA dataset



Slika 9.6. Konfuzijska matrica za XGBoost algoritam i korelacijski dataset

XGBClassifier algoritam postigao je najveću točnost u ovom radu, i to s datasetom u kojemu su uklonjene jako korelirane varijable. Također postignuta je i najmanja točnost u radu koristeći PCA predobrađeni dataset. Konfuzijske matrice potvrđuju navedeno, također se može vidjeti da *XGBoost* algoritam s korelacijskim datasetom samo dva puta pogrešno klasificirao ispravan rad stroja, što potvrđuje da ovaj algoritam jako dobro detektira greške u radu stroja.

Razlike između *MLPClassifier* i *XGBClassifier* algoritama očituju se u tome kako kojem algoritmu „odgovara“ predobrada podataka. Oba algoritma imaju vrlo dobre rezultate na potpunom datasetu, dok MLP algoritam u odnosu na originalni dataset pokazuje bolje rezultate s PCA predobradom a lošije rezultate s datasetom u kojemu su uklonjene jako korelirane varijable. XGBoost algoritam u odnosu na originalni dataset pokazuje bolje rezultate s datasetom u kojemu su uklonjene jako korelirane varijable a lošije rezultate s PCA predobrađenim datasetom.

10. ZAKLJUČAK

U ovome radu prikazane su mogućnosti primjene umjetne inteligencije u industriji za detekciju kvarova i dijagnostiku elektromotornih pogona. Umjetna inteligencija je iznimno važna grana znanosti u današnjem svijetu koji teži ka automatizaciji svakog dijela života. Algoritmi umjetne inteligencije su iznimno koristan alat za obradu velikih količina podataka, kao što su očitanja senzora postavljenih na elektromotorni pogon. Ti algoritmi mogu otkriti skrivene uzorke u tim podatcima i na taj način otkriti razne promjene u radu nekog stroja, te detektirati i dijagnosticirati razne kvarove i greške na stroju, te signalizirati kada je potrebno izvršiti održavanje na stroju.

Korišteni su algoritmi *XGBoost* i Višeslojni perceptron (*eng. MLP*), te su dostupni podatci predobrađivani metodom analize glavnih komponenti i uklanjanjem jako koreliranih podataka. Najbolje rezultate u detekciji i dijagnostici grešaka na stroju postigao je XGBoost algoritam s podatcima u kojima su uklonjene jako korelirane varijable, te je taj algoritam odabran kao najprikladniji za primjenu u ovom slučaju.

11. LITERATURA

[1] Gonzalez-Jimenez, D.; del-Olmo, J.; Poza, J.; Garramiola, F.; Madina, P. Data-Driven Fault Diagnosis for Electric Drives: A Review. Sensors 2021, 21, 4024.
doi: https://doi.org/10.3390/s21124024

[2] Car, Z.: Primjena umjetne inteligencije, predavanja i vježbe, Sveučilište u Rijeci, Tehnički fakultet, Rijeka,2021.

[3] STETCO, Adrian, et al. Machine learning methods for wind turbine condition monitoring: A review. Renewable energy, 2019, 133: 620-635.

[4] Dua,D. & Graff,C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[5] Martyna Bator, Alexander Dicks, Uwe Monks, & Volker Lohweg „Feature Extraction and Reduction Applied to Sensorless Drive Diagnosis“ Proc. 22. Workshop Computational Intelligence, Dortmund, 6.-7.12.2012.

[6] „A Step-by-Step Explanation of Principal Component Analysis (PCA)“, s Interneta, https://builtin.com/data-science/step-step-explanation-principal-component-analysis , 14. veljače 2022.

[7] „How to Create a Correlation Matrix using Pandas“, s Interneta, https://datatofish.com/correlation-matrix-pandas/ , 4. Rujna 2022.

[8] „How to drop out highly correlated features in Python?“, s Interneta, https://www.projectpro.io/recipes/drop-out-highly-correlated-features-in-python , 4. Rujna 2022.

[9] „A Beginner's Guide to Neural Networks and Deep Learning“, s Interneta, http://wiki.pathmind.com/neural-network , 4. Rujna 2022.

[10] „[Tutorial] Uvod u umjetnu inteligenciju, UNM“, s Interneta, https://forum.bug.hr/forum/topic/programiranje/tutorial-uvod-umjetnu-inteligenciju-unm/272724.aspx , 5. rujna 2022.

[11] Dalbelo-Bašić, B.: Stabla odluke, bilješke za predavanja, Sveučilište u Zagrabu, Fakultet elektrotehnike i računarstva, Zagreb, 2002.

[12] QUINLAN, J.. Ross . Induction of decision trees. *Machine learning*, 1986, 1.1: 81-106.

[13] „A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning“, s Interneta, https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/ , 4. Rujna 2022.

[14] „Extending Machine Learning Algorithms – AdaBoost Classifier“, s Interneta, https://www.youtube.com/watch?v=BoGNyWW9-mE , 16. rujna 2022.

[15] „Quick Introduction to Boosting Algorithms in Machine Learning“, s Interneta, https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/ , 4. Rujna 2022.

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U ovome diplomskom radu obrađene su metode dijagnostike elektromotornih pogona korištenjem umjetne inteligencije. Predstavljeni su i opisani algoritmi *XGBoost* i *MLPClassifier* koji su se primjenjivali za dijagnostiku u ovom radu, te je opisana analiza glavnih komponenti (PCA) i metoda uklanjanja jako koreliranih podataka kao metode koje se koriste za predobradu podataka. Uspoređeni su rezultati korištenjem oba algoritma bez predobrade i sa predobradom podataka, te je *XGBoost* algoritam s uklonjenim jako koreliranim podatcima odabran kao najbolji alat za dijagnostiku elektromotornih pogona metodama umjetne inteligencije u ovome slučaju.

Ključne riječi: UI, XGBoost, MLPClassifier, PCA, dijagnostika, inteligencija, korelacija.

ABSTRACT

This master thesis describes methods of electric drive fault diagnosis using artificial intelligence. XGBoost and MLPClassifier algorithms are described and used in this thesis, alongside Primary Component Analysis (PCA) and removal of highly correlated data as data preprocessing methods. Described algorithms are tested using original data and preprocessed data, the results are compared. XGBoost classification using preprocessed data via the removal of highly correlated data is deemed the best for fault diagnosis in this case.

Keywords: AI, XGBoost, MLPClassifier, PCA, diagnosis, intelligence, correlation.

DODATAK A – Tablica statističke analize

Redni br.	1	2	3	4	5	6
MIN	-0,013721	-0,0054144	-0,01358	-0,012787	-0,0083559	-0,0097413
MAX	0,0057836	0,0045253	0,0052377	0,0014531	0,00082451	0,0027536
Srednja vr.	-3,33285E-06	1,43965E-06	1,41201E-06	-1,31281E-06	1,35124E-06	-2,65448E-07
St. devijacija	7,17109E-05	5,55543E-05	0,000235301	6,25957E-05	5,66094E-05	0,000226191
Varijanca	5,14236E-09	3,08623E-09	5,53656E-08	3,91816E-09	3,20457E-09	5,11614E-08

Redni br.	7	8	9	10	11	12
MIN	-0,13989	-0,13594	-0,13086	-0,21864	-0,2186	-0,21863
MAX	0,069125	0,06913	0,069131	0,35258	0,35256	0,35263
Srednja vr.	0,001914585	0,001913146	0,001911733	-0,011897314	-0,011898658	-0,0118984
St. devijacija	0,036467597	0,03646549	0,036469794	0,066482331	0,066479902	0,066477078
Varijanca	0,001329863	0,001329709	0,001330023	0,004419825	0,004419502	0,004419126

Redni br.	13	14	15	16	17	18
MIN	0,00075094	0,00018844	0,0003542	0,00074448	0,0001889	0,00035701
MAX	0,13657	0,051543	0,10393	0,10877	0,064764	0,07853
Srednja vr.	0,001876354	0,001083408	0,003091717	0,001866515	0,00107748	0,003076336
St. devijacija	0,001019255	0,000667064	0,002194835	0,000949646	0,000656381	0,002130701
Varijanca	1,03886E-06	4,44967E-07	4,81722E-06	9,01812E-07	4,30829E-07	4,53981E-06

Redni br.	19	20	21	22	23	24
MIN	0,79764	0,79764	0,79763	0,7984	0,7984	0,79839
MAX	2,377	2,3769	2,3758	2,3728	2,3726	2,3715
Srednja vr.	1,618318873	1,618258029	1,617787041	1,617780067	1,617722925	1,617254983
St. devijacija	0,399767255	0,399740064	0,399591333	0,39815118	0,398125254	0,397964891
Varijanca	0,159811127	0,159789388	0,159670505	0,158521653	0,158501009	0,158373348

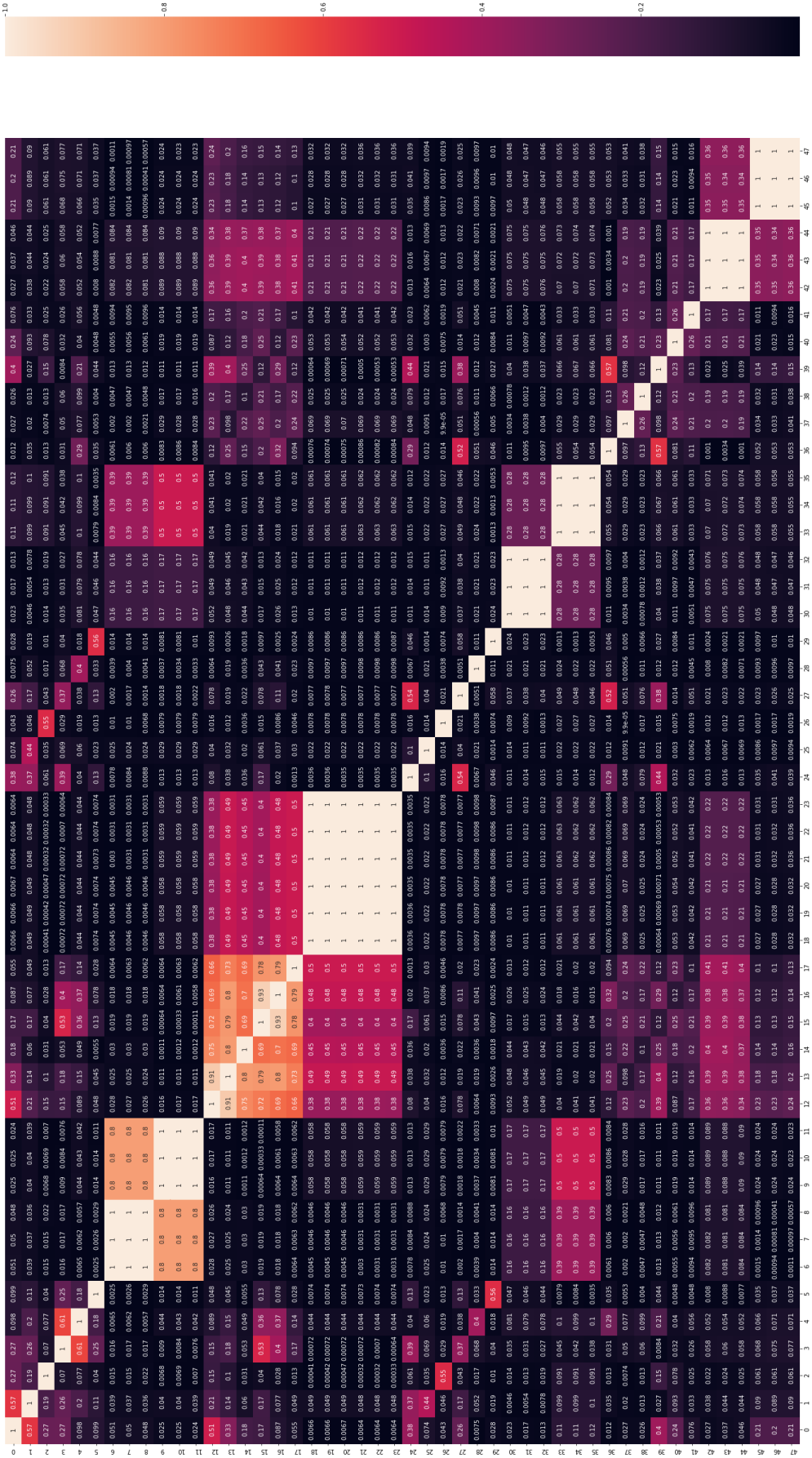
Redni br.	25	26	27	28	29	30
MIN	-15,796	-12,351	-7,959	-11,903	-12,508	-9,9766
MAX	28,285	12,437	9,5803	18,294	10,977	8,764
Srednja vr.	0,001908913	0,008798757	-0,003464656	-0,000157257	0,012089069	-0,009958415
St. devijacija	0,196070838	0,778815283	0,873891799	0,175249018	0,763320791	0,858767813
Varijanca	0,038443116	0,606542878	0,763673825	0,030711693	0,582648672	0,737469552

Redni br.	31	32	33	34	35	36
MIN	-0,050235	-0,051891	-0,052791	-0,33771	-0,3377	-0,33775
MAX	0,086378	0,086457	0,086553	0,19482	0,1902	0,18503
Srednja vr.	1,62776E-05	1,42048E-05	1,92158E-05	-3,47409E-05	-3,76187E-05	-3,1624E-05
St. devijacija	0,007797009	0,007791915	0,007794203	0,009885925	0,009876799	0,009870891
Varijanca	6,07923E-05	6,07129E-05	6,07486E-05	9,77298E-05	9,75495E-05	9,74328E-05

Redni br.	37	38	39	40	41	42
MIN	-0,9123	-0,61802	0,52218	-0,90235	-0,59683	0,32066
MAX	4015,4	312,52	265,33	3670,8	889,93	153,15
Srednja vr.	-0,46327633	7,447186317	8,406765045	-0,3977574	7,293781051	8,273771892
St. devijacija	20,39585467	12,24076521	6,897301166	25,01872797	12,45178148	6,565952073
Varijanca	415,983778	149,8337721	47,57195029	625,9260511	155,044212	43,11098979

Redni br.	43	44	45	46	47	48
MIN	-1,5255	-1,5262	-1,5237	-1,5214	-1,5232	-1,5213
MAX	-1,4576	-1,4561	-1,4555	-1,3372	-1,3372	-1,3371
Srednja vr.	-1,50088736	-1,50091155	-1,50080454	-1,49777144	-1,49779355	-1,49768563
St. devijacija	0,003657336	0,003667518	0,003631627	0,003162956	0,003163223	0,003174779
Varijanca	1,33759E-05	1,34505E-05	1,31885E-05	1,00041E-05	1,00058E-05	1,0079E-05

DODATAK B – Korelacijska matrica



DODATAK C – Python kod za gridsearch

MLPClassifier-puni dataset

```
import numpy as np
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
np.seterr(all="ignore")
warnings.filterwarnings('ignore')
#import tensorflow as tf
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D

from sklearn.model_selection import KFold, GridSearchCV
from sklearn.neural_network import MLPClassifier
import joblib

import pickle
import uuid


dataset = open("Sensorless_drive_diagnosis.txt")

X = []
Y = []

for ln in dataset.readlines():
    ct = 0
    x = []
    for dt in ln.split():
        ct = ct+1
        if ct == 49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x = np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X = np.array(X)
Y = np.array(Y)
descriptor="mlp_all"
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'hidden_layer_sizes':  [(40,40,40,40,40), (40,40,40,40),
                                        (20,20,20,20,20), (20,20,20,20),
                                        (20,40,100,40,20), (20,40,100),
(20,40,20), (40,100,20)],
                'activation': ['relu','identity','logistic','tanh'],
                'solver': ['adam', 'lbfgs'],
                'learning_rate':['constant','adaptive','invscaling'],
                'learning_rate_init': [0.1,0.01,0.5, 0.00001],
                'alpha': [0.001,0.002,0.00205, 0.003],
                'max_iter': [500]}]


model = GridSearchCV(MLPClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
model.fit(X,Y)
```

```
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"Fitting
DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n")

means1 = model.cv_results_['mean_test_score']
stds1 = model.cv_results_['std_test_score']
# means2 = model.cv_results_['mean_test_average_precision']
# stds2 = model.cv_results_['std_test_average_precision']

uuid_=uuid.uuid4()
file = open(descriptor+"-"+str(uuid_)+"-results.txt", 'w')
file.write("Data for: "+descriptor+"\n")
for mean1, std1, params, rank in zip(means1, stds1,model.cv_results_['params'],
model.cv_results_['rank_test_score']):

file.write("**********************************************************************
**********************************************************************\n")
        file.write("ACC: %0.20f (+/-%0.020f) for %r and rank %r" % (mean1, std1 *
2, params, rank)+"\n")
        if rank==1:
            best_params = params
            best_mean = mean1
            best_std = std1
file.write("**********************************************************************
**********************************************************************\n")
file.write("Best parameters are %r with accuracy of %0.20f (+/-%0.020f)" %
(best_params, best_mean, best_std * 2)+"\n")
file.close()
model_name = descriptor+"-"+str(uuid_)
print("Saving models...")
with open(model_name+'.pickle', 'wb') as f:
    pickle.dump(model, f)
from joblib import dump, load
joblib.dump(model.best_estimator_, model_name+".joblib")
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+
"\n")
```

MLPClassifier-PCA dataset

```
"""
Created on Mon Aug 22 10:06:59 2022

@author: tladi
"""

import numpy as np
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
np.seterr(all="ignore")
warnings.filterwarnings('ignore')
#import tensorflow as tf
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D

from sklearn.model_selection import KFold, GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
import joblib

import pickle
import uuid


dataset = open("Sensorless_drive_diagnosis.txt")

X = []
Y = []

for ln in dataset.readlines():
    ct = 0
    x = []
    for dt in ln.split():
        ct = ct+1
        if ct == 49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x = np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X = np.array(X)
Y = np.array(Y)


# skaliranje podataka
scaler = StandardScaler()
scaler.fit(X)

X = scaler.transform(X)


# pca

pca = PCA(.97)

pca.fit(X)

X = pca.transform(X)

descriptor="mlp_PCA"
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'hidden_layer_sizes':  [(40,40,40,40,40), (40,40,40,40),
                                        (20,20,20,20,20), (20,20,20,20),
                                        (20,40,100,40,20), (20,40,100),
(20,40,20), (40,100,20)],
                'activation': ['relu','identity','logistic','tanh'],
                'solver': ['adam', 'lbfgs'],
                'learning_rate':['constant','adaptive','invscaling'],
                'learning_rate_init': [0.1,0.01,0.5, 0.00001],
                'alpha': [0.01,0.02,0.0205, 0.0001],
                'max_iter': [500]}]


model = GridSearchCV(MLPClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
model.fit(X,Y)
```

```
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"Fitting
DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n")

means1 = model.cv_results_['mean_test_score']
stds1 = model.cv_results_['std_test_score']
# means2 = model.cv_results_['mean_test_average_precision']
# stds2 = model.cv_results_['std_test_average_precision']

uuid_=uuid.uuid4()
file = open(descriptor+"-"+str(uuid_)+"-results.txt", 'w')
file.write("Data for: "+descriptor+"\n")
for mean1, std1, params, rank in zip(means1, stds1,model.cv_results_['params'],
model.cv_results_['rank_test_score']):

file.write("**********************************************************************
**********************************************************************\n")
        file.write("ACC: %0.20f (+/-%0.020f) for %r and rank %r" % (mean1, std1 *
2, params, rank)+"\n")
        if rank==1:
            best_params = params
            best_mean = mean1
            best_std = std1
file.write("**********************************************************************
**********************************************************************\n")
file.write("Best parameters are %r with accuracy of %0.20f (+/-%0.020f)" %
(best_params, best_mean, best_std * 2)+"\n")
file.close()
model_name = descriptor+"-"+str(uuid_)
print("Saving models...")
with open(model_name+'.pickle', 'wb') as f:
    pickle.dump(model, f)
from joblib import dump, load
joblib.dump(model.best_estimator_, model_name+".joblib")
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+
"\n")
```

MLPClassifier-Korelacijski dataset

```
"""
Created on Mon Aug 22 10:38:40 2022

@author: tladi
"""

import numpy as np
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
np.seterr(all="ignore")
warnings.filterwarnings('ignore')
#import tensorflow as tf
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D

from sklearn.model_selection import KFold, GridSearchCV
from sklearn.neural_network import MLPClassifier
import joblib
import pandas as pd
import pickle
```

```
import uuid


dataset = open("Sensorless_drive_diagnosis.txt")

X_org= []
Y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X_org.append(x)

X_org=np.array(X_org)
Y=np.array(Y)

df= pd.DataFrame(X_org)
corrMatrix = df.corr().abs()


upper_tri =
corrMatrix.where(np.triu(np.ones(corrMatrix.shape),k=1).astype(np.bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] >
0.95)]
X = df.drop(df.columns[to_drop], axis=1)


descriptor="mlp_corr"
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'hidden_layer_sizes':  [(40,40,40,40,40), (40,40,40,40),
                                        (20,20,20,20,20), (20,20,20,20),
                                        (20,40,100,40,20), (20,40,100),
(20,40,20), (40,100,20)],
                'activation': ['relu','identity','logistic','tanh'],
                'solver': ['adam', 'lbfgs'],
                'learning_rate':['constant','adaptive','invscaling'],
                'learning_rate_init': [0.1,0.01,0.5, 0.00001],
                'alpha': [0.01,0.02,0.0205, 0.0001],
                'max_iter': [500]}]

model = GridSearchCV(MLPClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
model.fit(X,Y)

print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"Fitting
DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n")

means1 = model.cv_results_['mean_test_score']
stds1 = model.cv_results_['std_test_score']
# means2 = model.cv_results_['mean_test_average_precision']
# stds2 = model.cv_results_['std_test_average_precision']
```

```
uuid_=uuid.uuid4()
file = open(descriptor+"-"+str(uuid_)+"-results.txt", 'w')
file.write("Data for: "+descriptor+"\n")
for mean1, std1, params, rank in zip(means1, stds1,model.cv_results_['params'],
model.cv_results_['rank_test_score']):

file.write("**********************************************************************
**********************************************************************\n")
        file.write("ACC: %0.20f (+/-%0.020f) for %r and rank %r" % (mean1, std1 *
2, params, rank)+"\n")
        if rank==1:
            best_params = params
            best_mean = mean1
            best_std = std1
file.write("**********************************************************************
**********************************************************************\n")
file.write("Best parameters are %r with accuracy of %0.20f (+/-%0.020f)" %
(best_params, best_mean, best_std * 2)+"\n")
file.close()
model_name = descriptor+"-"+str(uuid_)
print("Saving models...")
with open(model_name+'.pickle', 'wb') as f:
    pickle.dump(model, f)
from joblib import dump, load
joblib.dump(model.best_estimator_, model_name+".joblib")
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+
"\n")
```

XGBClassifier-puni dataset

```
"""
Created on Mon Aug 22 19:19:46 2022

@author: tladi
"""

import numpy as np
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
np.seterr(all="ignore")
warnings.filterwarnings('ignore')
#import tensorflow as tf
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D

from sklearn.model_selection import KFold, GridSearchCV
from xgboost import XGBClassifier
import joblib
import pickle
import uuid



dataset = open("Sensorless_drive_diagnosis.txt")

X = []
Y = []

for ln in dataset.readlines():
```

```
    ct = 0
    x = []
    for dt in ln.split():
        ct = ct+1
        if ct == 49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x = np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X = np.array(X)
Y = np.array(Y)
Y=Y-1;

descriptor="XGboost_sve"
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'booster':  ['gbtree', 'dart'],
                'learning_rate': [0.05, 0.1, 0.2],
                'max_depth': [9, 10, 11],
                'min_child_weight': [7, 8, 9],
                'colsample_bylevel':[0.9, 1],
                'colsample_bynode':[ 0.9, 1],
                'colsample_bytree':[0.9, 1],
                'reg_alpha': [0, 1],
                'reg_lambda': [1, 2],
                'use_label_encoder':[False],
                'n_estimators':[600]}]



model = GridSearchCV(XGBClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
model.fit(X,Y)

print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"Fitting
DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n")

means1 = model.cv_results_['mean_test_score']
stds1 = model.cv_results_['std_test_score']
# means2 = model.cv_results_['mean_test_average_precision']
# stds2 = model.cv_results_['std_test_average_precision']

uuid_=uuid.uuid4()
file = open(descriptor+"-"+str(uuid_)+"-results.txt", 'w')
file.write("Data for: "+descriptor+"\n")
for mean1, std1, params, rank in zip(means1, stds1,model.cv_results_['params'],
model.cv_results_['rank_test_score']):

file.write("**********************************************************************
**************************************************************\n")
        file.write("ACC: %0.20f (+/-%0.020f) for %r and rank %r" % (mean1, std1 *
2, params, rank)+"\n")
        if rank==1:
            best_params = params
            best_mean = mean1
            best_std = std1
file.write("**********************************************************************
**************************************************************\n")
```

```
file.write("Best parameters are %r with accuracy of %0.20f (+/-%0.020f)" %
(best_params, best_mean, best_std * 2)+"\n")
file.close()
model_name = descriptor+"-"+str(uuid_)
print("Saving models...")
with open(model_name+'.pickle', 'wb') as f:
    pickle.dump(model, f)
from joblib import dump, load
joblib.dump(model.best_estimator_, model_name+".joblib")
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+
"\n")
```

XGBClassifier-PCA dataset

```
"""
Created on Mon Aug 22 20:13:29 2022

@author: tladi
"""

import numpy as np
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
np.seterr(all="ignore")
warnings.filterwarnings('ignore')
#import tensorflow as tf
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D

from sklearn.model_selection import KFold, GridSearchCV
from xgboost import XGBClassifier
import joblib
import pickle
import uuid
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA


dataset = open("Sensorless_drive_diagnosis.txt")

X = []
Y = []

for ln in dataset.readlines():
    ct = 0
    x = []
    for dt in ln.split():
        ct = ct+1
        if ct == 49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x = np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X = np.array(X)
Y = np.array(Y)
Y=Y-1;
```

```
# skaliranje podataka
scaler = StandardScaler()
scaler.fit(X)

X = scaler.transform(X)


# pca

pca = PCA(.97)

pca.fit(X)

X = pca.transform(X)

descriptor="XGboost_PCA"
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'booster':  ['gbtree', 'dart'],
                'learning_rate': [0.05, 0.1, 0.2],
                'max_depth': [9, 10, 11],
                'min_child_weight': [7, 8, 9],
                'colsample_bylevel':[0.9, 1],
                'colsample_bynode':[ 0.9, 1],
                'colsample_bytree':[0.9, 1],
                'reg_alpha': [0, 1],
                'reg_lambda': [1, 2],
                'use_label_encoder':[False],
                'n_estimators':[600]}]

model = GridSearchCV(XGBClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
model.fit(X,Y)

print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"Fitting
DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n")

means1 = model.cv_results_['mean_test_score']
stds1 = model.cv_results_['std_test_score']
# means2 = model.cv_results_['mean_test_average_precision']
# stds2 = model.cv_results_['std_test_average_precision']

uuid_=uuid.uuid4()
file = open(descriptor+"-"+str(uuid_)+"-results.txt", 'w')
file.write("Data for: "+descriptor+"\n")
for mean1, std1, params, rank in zip(means1, stds1,model.cv_results_['params'],
model.cv_results_['rank_test_score']):

file.write("**********************************************************************
**************************************************************\n")
        file.write("ACC: %0.20f (+/-%0.020f) for %r and rank %r" % (mean1, std1 *
2, params, rank)+"\n")
        if rank==1:
            best_params = params
            best_mean = mean1
            best_std = std1
file.write("**********************************************************************
**************************************************************\n")
```

```
file.write("Best parameters are %r with accuracy of %0.20f (+/-%0.020f)" %
(best_params, best_mean, best_std * 2)+"\n")
file.close()
model_name = descriptor+"-"+str(uuid_)
print("Saving models...")
with open(model_name+'.pickle', 'wb') as f:
    pickle.dump(model, f)
from joblib import dump, load
joblib.dump(model.best_estimator_, model_name+".joblib")
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+
"\n")
```

XGBClassifier-Korelacijski dataset

```
"""
Created on Mon Aug 22 20:16:19 2022

@author: tladi
"""

import numpy as np
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
np.seterr(all="ignore")
warnings.filterwarnings('ignore')
#import tensorflow as tf
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D,
MaxPooling2D

from sklearn.model_selection import KFold, GridSearchCV
from xgboost import XGBClassifier
import joblib
import pickle
import uuid
import pandas as pd



dataset = open("Sensorless_drive_diagnosis.txt")

X_org= []
Y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X_org.append(x)

X_org=np.array(X_org)
Y=np.array(Y)
Y=Y-1;
```

```
df= pd.DataFrame(X_org)

corrMatrix = df.corr().abs()


upper_tri =
corrMatrix.where(np.triu(np.ones(corrMatrix.shape),k=1).astype(np.bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] >
0.95)]
X = df.drop(df.columns[to_drop], axis=1)


descriptor="XGboost_corr"
kfold = KFold(n_splits=10, shuffle=True)

params_dict = [{'booster':  ['gbtree', 'dart'],
                'learning_rate': [0.05, 0.1, 0.2],
                'max_depth': [9, 10, 11],
                'min_child_weight': [7, 8, 9],
                'colsample_bylevel':[0.9, 1],
                'colsample_bynode':[ 0.9, 1],
                'colsample_bytree':[0.9, 1],
                'reg_alpha': [0, 1],
                'reg_lambda': [1, 2],
                'use_label_encoder':[False],
                'n_estimators':[600]}]

scores = ['accuracy', 'average_precision']
model = GridSearchCV(XGBClassifier(), params_dict, cv=kfold, n_jobs=-1,
scoring='accuracy', verbose=10, refit=True)
model.fit(X,Y)

print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"Fitting
DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n")

means1 = model.cv_results_['mean_test_score']
stds1 = model.cv_results_['std_test_score']
# means2 = model.cv_results_['mean_test_average_precision']
# stds2 = model.cv_results_['std_test_average_precision']

uuid_=uuid.uuid4()
file = open(descriptor+"-"+str(uuid_)+"-results.txt", 'w')
file.write("Data for: "+descriptor+"\n")
for mean1, std1, params, rank in zip(means1, stds1,model.cv_results_['params'],
model.cv_results_['rank_test_score']):

file.write("**********************************************************************
**********************************************************************\n")
        file.write("ACC: %0.20f (+/-%0.020f) for %r and rank %r" % (mean1, std1 *
2, params, rank)+"\n")
        if rank==1:
            best_params = params
            best_mean = mean1
            best_std = std1
file.write("**********************************************************************
**********************************************************************\n")
file.write("Best parameters are %r with accuracy of %0.20f (+/-%0.020f)" %
(best_params, best_mean, best_std * 2)+"\n")
file.close()
model_name = descriptor+"-"+str(uuid_)
```

```
print("Saving models...")
with open(model_name+'.pickle', 'wb') as f:
    pickle.dump(model, f)
from joblib import dump, load
joblib.dump(model.best_estimator_, model_name+".joblib")
print(50*"*"+"\n"+50*"*"+"\n"+50*"*"+"\n"+"DONE\n"+50*"*"+"\n"+50*"*"+"\n"+50*"*"+
"\n")
```

DODATAK D – Python kod za MLP klasifikaciju

Puni dataset

```
"""
Created on Sun Aug 22 17:26:08 2021

@author: tladi
"""

import numpy as np
from matplotlib import pyplot as plt
from sklearn.neural_network import MLPClassifier
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

dataset = open("Sensorless_drive_diagnosis.txt")

X = []
y = []

for ln in dataset.readlines():
    ct = 0
    x = []
    for dt in ln.split():
        ct = ct+1
        if ct == 49:
            y.append(int(dt))
        else:
            x.append(float(dt))
    x = np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X = np.array(X)
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=0)


clf = MLPClassifier(random_state=0, alpha=0.0001, solver='adam',
                    hidden_layer_sizes=(40,40,40,40),
                    activation='logistic', verbose='True',
                    learning_rate='constant', max_iter=500,
                    learning_rate_init=0.01)


# Treniranje klasifikatora s podacima za treniranje
clf.fit(X_train, y_train)


loss = clf.loss_
print("Loss: "+str(loss))
accuracy = clf.score(X_test, y_test)
print("Accuracy: "+str(accuracy))
n_iter = clf.n_iter_
print("Number of iterations: "+str(n_iter))
```

```
loss_curve = clf.loss_curve_
plt.figure(figsize=(10, 8))
plt.rcParams["font.family"] = "Times New Roman"
SMALL_SIZE = 20
MEDIUM_SIZE = 22
BIGGER_SIZE = 24
plt.rcParams["legend.loc"] = "upper left"
plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
plt.title('Training loss')
plt.ylabel('Gubici')
plt.xlabel('Broj iteracije')
plt.plot(loss_curve)
plt.grid(True)
plt.show()


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Točno')
    plt.xlabel('Predviđeno')
    plt.show()
    plt.rcParams["font.family"] = "Times New Roman"
    SMALL_SIZE = 20
    MEDIUM_SIZE = 22
    BIGGER_SIZE = 24
    plt.rcParams["legend.loc"] = "upper left"
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

```
predicted = clf.predict(X_test)
cm = confusion_matrix(y_test, predicted)
c = np.array(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11'])
plot_confusion_matrix(cm, classes=c,
                      title='Konfuzijska matrica')
accuracy_score(y_test, predicted, normalize=True)
```

PCA dataset

```
"""
Created on Tue Jan  4 16:40:17 2022

@author: tladi
"""

import numpy as np
from matplotlib import pyplot as plt
from sklearn.neural_network import MLPClassifier
import itertools
#from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import  classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

dataset=open("Sensorless_drive_diagnosis.txt")

X= []
y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X=np.array(X)
y=np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=0)
# skaliranje podataka
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# pca

pca = PCA(.97)
```

```
pca.fit(X_train)

X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

#Unos parametara klasifikatora
#clf = MLPClassifier(random_state=0, alpha=0.00205, solver = 'adam',
hidden_layer_sizes=(20,40,100,40,20), activation = 'tanh', verbose = 'True',
learning_rate = 'adaptive',max_iter = 500)
#clf = MLPClassifier(random_state=0, alpha=0.0205, solver = 'lbfgs',
hidden_layer_sizes=(20,40,100,40,20), activation = 'tanh', learning_rate =
'adaptive',max_iter = 500)
clf = MLPClassifier(random_state=0, alpha=0.0205, solver='lbfgs',
hidden_layer_sizes=(20,40,100,40,20), activation='tanh', verbose = 1,
learning_rate='adaptive', max_iter=500, learning_rate_init=0.0001)

#Treniranje klasifikatora s podacima za treniranje
clf.fit(X_train,y_train)

accuracy=clf.score(X_test, y_test)
print("Accuracy: "+str(accuracy))

loss=clf.loss_
print("Loss: "+str(loss))

n_iter=clf.n_iter_
print("Number of iterations: "+str(n_iter))
# loss_curve=clf.loss_curve_
# plt.figure(figsize=(10, 8))
plt.rcParams["font.family"] = "Times New Roman"
SMALL_SIZE = 20
MEDIUM_SIZE = 22
BIGGER_SIZE = 24
plt.rcParams["legend.loc"] = "upper left"
plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
# plt.title('Training loss')
# plt.ylabel('Gubici')
# plt.xlabel('Broj iteracije')
# plt.plot(loss_curve)
# plt.grid(True)



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
```

```
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Točno')
    plt.xlabel('Predviđeno')
    plt.show()
    plt.rcParams["font.family"] = "Times New Roman"
    SMALL_SIZE = 20
    MEDIUM_SIZE = 22
    BIGGER_SIZE = 24
    plt.rcParams["legend.loc"] = "upper left"
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title

train_prediction= clf.predict(X_train)

predicted = clf.predict(X_test)
cm = confusion_matrix(y_test,predicted)
c=np.array(['1','2','3','4','5','6','7','8','9','10','11'])
plot_confusion_matrix(cm, classes=c,\
                      title='Konfuzijska matrica')
train_accuracy=accuracy_score(y_train, train_prediction)
print("Train Accuracy: "+str(train_accuracy))

test_accuracy=accuracy_score(y_test, predicted)
print("Test Accuracy: "+str(test_accuracy))
cr= classification_report(y_test, predicted)
print(cr)
```

Korelacijski dataset

```
"""
Created on Tue Jan  4 16:14:03 2022

@author: tladi
"""

import numpy as np
from matplotlib import pyplot as plt
from sklearn.neural_network import MLPClassifier
import itertools
import pandas as pd
import seaborn as sn
from sklearn.metrics import  classification_report, accuracy_score

from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix
```

```
dataset=open("Sensorless_drive_diagnosis.txt")

X_org= []
Y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            Y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X_org.append(x)

X_org=np.array(X_org)
Y=np.array(Y)

df= pd.DataFrame(X_org)
print(); print(df.head())
corrMatrix = df.corr().abs()
plt.figure(figsize=(48,24))
sn.heatmap(corrMatrix, annot=True)
plt.show()

upper_tri =
corrMatrix.where(np.triu(np.ones(corrMatrix.shape),k=1).astype(np.bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] >
0.95)]
X = df.drop(df.columns[to_drop], axis=1)
print(); print(X.head())

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,
random_state=0)

#Unos parametara klasifikatora
#clf = MLPClassifier(random_state=0, alpha=0.00205, solver = 'adam',
hidden_layer_sizes=(20,40,100,40,20), activation = 'tanh', verbose = 'True',
learning_rate = 'adaptive',max_iter = 500)
clf = MLPClassifier(random_state=0, alpha=0.0001, solver='adam',
                    hidden_layer_sizes=(40,100,20),
                    activation='logistic', verbose='True',
                    learning_rate='invscaling', max_iter=500,
                    learning_rate_init=0.01)

#Treniranje klasifikatora s podacima za treniranje
clf.fit(X_train,y_train)


loss=clf.loss_
print("Loss: "+str(loss))
accuracy=clf.score(X_test, y_test)
print("Accuracy: "+str(accuracy))
n_iter=clf.n_iter_
print("Number of iterations: "+str(n_iter))
loss_curve=clf.loss_curve_
plt.figure(figsize=(10, 8))
```

```
plt.rcParams["font.family"] = "Times New Roman"
SMALL_SIZE = 20
MEDIUM_SIZE = 22
BIGGER_SIZE = 24
plt.rcParams["legend.loc"] = "upper left"
plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
plt.title('Training loss')
plt.ylabel('Gubici')
plt.xlabel('Broj iteracije')
plt.plot(loss_curve)
plt.grid(True)



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Točno')
    plt.xlabel('Predviđeno')
    plt.show()
    plt.rcParams["font.family"] = "Times New Roman"
    SMALL_SIZE = 20
    MEDIUM_SIZE = 22
    BIGGER_SIZE = 24
    plt.rcParams["legend.loc"] = "upper left"
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title


train_prediction= clf.predict(X_train)
```

```
predicted = clf.predict(X_test)
cm = confusion_matrix(y_test,predicted)
c=np.array(['1','2','3','4','5','6','7','8','9','10','11'])
plot_confusion_matrix(cm, classes=c,\
                      title='Konfuzijska matrica')
train_accuracy=accuracy_score(y_train, train_prediction)
print("Train Accuracy: "+str(train_accuracy))

test_accuracy=accuracy_score(y_test, predicted)
print("Test Accuracy: "+str(test_accuracy))
cr= classification_report(y_test, predicted)
print(cr)
```

DODATAK E – Python kod za XGBoost klasifikaciju

Puni dataset

```
"""
Created on Tue Jun  7 21:50:22 2022

@author: tladi
"""

import numpy as np
from matplotlib import pyplot as plt

import itertools
from xgboost import XGBClassifier
from sklearn.metrics import  classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix

dataset=open("Sensorless_drive_diagnosis.txt")

X= []
y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X=np.array(X)
y=np.array(y)
y=y-1;

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

#Unos parametara klasifikatora
clf1= XGBClassifier( colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, learning_rate=0.05,
              max_depth=10, min_child_weight=8,
              n_estimators=600,
              random_state=0, reg_alpha=0, reg_lambda=1,
              subsample=0.5,
              verbosity=0)


#Treniranje klasifikatora s podacima za treniranje
clf1.fit(X_train,y_train)




def plot_confusion_matrix(cm, classes,
```

```
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Točno')
    plt.xlabel('Predviđeno')
    plt.show()
    plt.rcParams["font.family"] = "Times New Roman"
    SMALL_SIZE = 20
    MEDIUM_SIZE = 22
    BIGGER_SIZE = 24
    plt.rcParams["legend.loc"] = "upper left"
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title


train_prediction= clf1.predict(X_train)

predicted = clf1.predict(X_test)
cm = confusion_matrix(y_test,predicted)
c=np.array(['1','2','3','4','5','6','7','8','9','10','11'])
plot_confusion_matrix(cm, classes=c,\
                      title='Konfuzijska matrica')
train_accuracy=accuracy_score(y_train, train_prediction)
print("Train Accuracy: "+str(train_accuracy))

test_accuracy=accuracy_score(y_test, predicted)
print("Test Accuracy: "+str(test_accuracy))
cr= classification_report(y_test, predicted)
print(cr)
```

PCA dataset

```
"""
Created on Tue Jun  7 22:04:25 2022

@author: tladi
"""

import numpy as np
from matplotlib import pyplot as plt
from sklearn.neural_network import MLPClassifier
import itertools
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix
from sklearn.metrics import  classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

dataset=open("Sensorless_drive_diagnosis.txt")

X= []
y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X=np.array(X)
y=np.array(y)
y=y-1;

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=0)
# skaliranje podataka
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# pca

pca = PCA(.97)

pca.fit(X_train)

X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

#Unos parametara klasifikatora
```

```
clf1= XGBClassifier( colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, learning_rate=0.05,
              max_depth=10, min_child_weight=8,
              n_estimators=600,
              random_state=0, reg_alpha=0, reg_lambda=1,
              subsample=0.5,
              verbosity=None)


#Treniranje klasifikatora s podacima za treniranje
clf1.fit(X_train,y_train)




def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Točno')
    plt.xlabel('Predviđeno')
    plt.show()
    plt.rcParams["font.family"] = "Times New Roman"
    SMALL_SIZE = 20
    MEDIUM_SIZE = 22
    BIGGER_SIZE = 24
    plt.rcParams["legend.loc"] = "upper left"
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title

train_prediction= clf1.predict(X_train)

predicted = clf1.predict(X_test)
cm = confusion_matrix(y_test,predicted)
c=np.array(['1','2','3','4','5','6','7','8','9','10','11'])
```

```
plot_confusion_matrix(cm, classes=c,\
                      title='Konfuzijska matrica')
train_accuracy=accuracy_score(y_train, train_prediction)
print("Train Accuracy: "+str(train_accuracy))

test_accuracy=accuracy_score(y_test, predicted)
print("Test Accuracy: "+str(test_accuracy))
cr= classification_report(y_test, predicted)
print(cr)
```

Korelacijski dataset

```
"""
Created on Wed Jun  8 23:07:09 2022

@author: tladi
"""

import numpy as np
from matplotlib import pyplot as plt
from sklearn.neural_network import MLPClassifier
import itertools
import pandas as pd
import seaborn as sn
from sklearn.metrics import  classification_report, accuracy_score
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix

dataset=open("Sensorless_drive_diagnosis.txt")

X= []
y= []

for ln in dataset.readlines():
    ct=0
    x=[]
    for dt in ln.split():
        ct=ct+1
        if ct==49:
            y.append(int(dt))
        else:
            x.append(float(dt))
    x=np.array(x)
    np.reshape(x, (1, 48))
    X.append(x)

X=np.array(X)
y=np.array(y)
y=y-1;

df= pd.DataFrame(X)
print(); print(df.head())
corrMatrix = df.corr().abs()
plt.figure(figsize=(48,24))
sn.heatmap(corrMatrix, annot=True)
plt.show()

upper_tri =
corrMatrix.where(np.triu(np.ones(corrMatrix.shape),k=1).astype(np.bool))
```

```
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] >
0.90)]
df1 = df.drop(df.columns[to_drop], axis=1)
print(); print(df1.head())

X_train, X_test, y_train, y_test = train_test_split(df1, y, test_size=0.25,
random_state=0)

clf1= XGBClassifier( colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, learning_rate=0.05,
              max_depth=10, min_child_weight=8,
              n_estimators=600,
              random_state=0, reg_alpha=0, reg_lambda=1,
              subsample=0.5,
              verbosity=None)


#Treniranje klasifikatora s podacima za treniranje
clf1.fit(X_train,y_train)




def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Točno')
    plt.xlabel('Predviđeno')
    plt.show()
    plt.rcParams["font.family"] = "Times New Roman"
    SMALL_SIZE = 20
    MEDIUM_SIZE = 22
    BIGGER_SIZE = 24
    plt.rcParams["legend.loc"] = "upper left"
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
```

```
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title

train_prediction= clf1.predict(X_train)

predicted = clf1.predict(X_test)
cm = confusion_matrix(y_test,predicted)
c=np.array(['1','2','3','4','5','6','7','8','9','10','11'])
plot_confusion_matrix(cm, classes=c,\
                      title='Konfuzijska matrica')
train_accuracy=accuracy_score(y_train, train_prediction)
print("Train Accuracy: "+str(train_accuracy))

test_accuracy=accuracy_score(y_test, predicted)
print("Test Accuracy: "+str(test_accuracy))
cr= classification_report(y_test, predicted)
print(cr)
```