

PROJEKTNI ZADATAK

(MLP kalsifikacija i Reinforcment learning)

***Predmet****:* Inteligentni sistemi

|  |  |
| --- | --- |
| **PROFESOR:** | **STUDENTI**: |
| dr. sci. Lejla Banjanović-Mehmedović,  vanr. prof. | Nura Efendić  Arnela Lisić  Azur Jusić |

TUZLA, 2024. god.

**Sadržaj**

[1. Zadatak 2 4](#_Toc1)

[1.1. Uvod 4](#_Toc2)

[1.2. Opis Problema 4](#_Toc3)

[1.3. Komponente 4](#_Toc4)

[1.3.1. Neuronska Mreža (QNetwork) 4](#_Toc5)

[1.3.2. Funkcija Nagrade (get\_reward) 5](#_Toc6)

[1.3.3. Petlja za obuku 5](#_Toc7)

[1.3.4. Kontrolni sistem 7](#_Toc8)

[1.4. Proces obuke 8](#_Toc9)

[1.4.1. Istraživanje-Eksploatacija 8](#_Toc10)

[1.4.2. Ažuriranje Q-Mreže 8](#_Toc11)

[1.5. Rezultati 9](#_Toc12)

[1.5.1. Ishod Obuke 9](#_Toc13)

[1.5.2. Ograničenja  12](#_Toc14)

[1.6. Zaključak 13](#_Toc15)

[2. Zadatak 1 14](#_Toc16)

[2.1. Uvod 14](#_Toc17)

[2.1.1. Kristalni sistemi i njihova svojstva 14](#_Toc18)

[2.2. Predprocesiranje podataka 15](#_Toc19)

[2.3. Podešavanje(tuning) modela a) 20](#_Toc20)

[2.4. Podešavanje (tuning) modela b) 27](#_Toc21)

[2.5. Rješenje c) 32](#_Toc22)

[2.6. Rješenje d) 38](#_Toc23)

[2.7. Dodatno podesavanje e) 39](#_Toc24)

**Popis slika:**

[Slika 1 Zavisnost ukupne nagrade od broja epizode za epsilon = 0.5 i gamma = 0.9 10](#_Toc1)

[Slika 2 Vrijednosti parametara po epizodi za prethodnu sliku 10](#_Toc2)

[Slika 3 Zavisnost ukupne nagrade od broja epizode za epsilon = 0.1 i gamma = 0.9 nakon 3000 epizoda treniranih sa parametron epsilon = 0.1 12](#_Toc3)

[Slika 4 Kolona ‘formula’ 15](#_Toc4)

[Slika 5 Konfuzijske matrice 16](#_Toc5)

[Slika 6 Konfuzijske matrice a) 26](#_Toc6)

[Slika 7 Očekivani izlazi a) 26](#_Toc7)

[Slika 8 Usporedba klasifikatora 27](#_Toc8)

[Slika 9 Konfuzijske matrice 31](#_Toc9)

[Slika 10 Očekivani izlazi 31](#_Toc10)

[Slika 11 Performanse 32](#_Toc11)

[Slika 12 Usporedba rezultata 38](#_Toc12)

[Slika 13 Konfuzijske matrice 38](#_Toc13)

[Slika 14 Ocekivani izlazi 38](#_Toc14)

[Slika 15 Usproedba d) 39](#_Toc15)

[Slika 16 Konfuzijske matrice e) 39](#_Toc16)

[Slika 17 Očekivani izlazi 40](#_Toc17)

[Slika 18 Usporedba klasifikatora 40](#_Toc18)

**Popis tabela**

[Tabela 1 Evaluacijska metrika a) 26](#_Toc1)

[Tabela 2 rezultati 32](#_Toc2)

[Tabela 3 Očekivani izlazi 37](#_Toc3)

[Tabela 4 rezultati c) 37](#_Toc4)

[Tabela 5 rjesenja d) 39](#_Toc5)

[Tabela 6 Rezultati 40](#_Toc6)

Popis kodova

[Kod 1 regulacija temperature 4](#_Toc1)

[Kod 2 Q network 4](#_Toc2)

[Kod 3 Obuka agenta 5](#_Toc3)

[Kod 4 Petlja za obuku 7](#_Toc4)

[Kod 5 Kontrolni sistem 7](#_Toc5)

[Kod 6 Eksploatacija 8](#_Toc6)

[Kod 7 Azuriranje 8](#_Toc7)

[Kod 8 20](#_Toc8)

[Kod 9 train\_test\_split 21](#_Toc9)

[Kod 10 Tuning a) 25](#_Toc10)

[Kod 14 Tuning b) 31](#_Toc11)

[Kod 15 37](#_Toc12)

# Zadatak 2

## Uvod

Sistem kontrole temperature prostorije koji je prikazan u priloženom kodu implementira pristup reinforcement learning (RL, učenje sa podrškom) kako bi se održala temperatura prostorije unutar željenog raspona. RL je grana mašinskog učenja gdje agent uči kako donositi odluke interakcijom s okolinom i primanjem povratnih informacija u obliku nagrada.

## Opis Problema

Glavni cilj sistema je regulirati temperaturu prostorije tako da bude unutar optimalnog raspona od 21 do 24 stepeni Celzijusa. To se postiže dinamičkim prilagođavanjem sistema grijanja na temelju trenutne temperature prostorije i  temperature van prostorije.

|  |
| --- |
| def is\_temperature\_optimal(room\_temperature):  return 21 <= room\_temperature <= 24 |

Kod 1 regulacija temperature

## Komponente

### Neuronska Mreža (QNetwork)

QNetwork je neuronska mreža implementirana pomoću PyTorcha. Prima trenutnu temperaturu prostorije i status grijanja kao ulaz te izlazne Q-vrijednosti za svaku moguću akciju. Arhitektura se sastoji od dva potpuno povezana sloja. Odabir broja neurona u prvom skrivenom sloju je ya veci broj epizoda(do 10000) uzet kao 128, a za manji broj epizoda (100) uzet da je 12.

|  |
| --- |
| class QNetwork(nn.Module):  def \_\_init\_\_(self, input\_size, output\_size):       super(QNetwork, self).\_\_init\_\_()       self.fc1 = nn.Linear(input\_size, 128)       self.fc2 = nn.Linear(128, output\_size) |

Kod 2 Q network

### Funkcija Nagrade (get\_reward)

Funkcija nagrade definira nagrade na temelju trenutnog stanja i akcije. Pozitivne nagrade dodjeljuju se za akcije koje vode optimalnom rasponu temperature, dok se negativne nagrade dodjeljuju inače. Ova funkcija igra ključnu ulogu u vođenju procesa učenja agenta.

|  |
| --- |
| def get\_reward(room\_temp, heating\_on):  def get\_reward(room\_temp, heating\_on):  if room\_temp < 21 and heating\_on==1:       return 10  elif room\_temp<21 and heating\_on==0:       return -10  if 21 <= room\_temp <= 22 and heating\_on==1:       return 10  elif 22<= room\_temp <= 24 and heating\_on==0:       return 10  elif 23<= room\_temp <= 24 and heating\_on ==1:       return 1  elif room\_temp > 24 and room\_temp<=25 and heating\_on ==1:       return -5  elif room\_temp >25 and heating\_on==0:       return 10  elif room\_temp > 25 and heating\_on ==1:       return -15  else:       return -5 |

Kod 3 Obuka agenta

### Petlja za obuku

Petlja za obuku prolazi kroz više epizoda, gdje svaka epizoda simulira okolinu s nasumičnim uvjetima temperature i stanjima grijanja. Tokom svake epizode, neuronska mreža se trenira kako bi ažurirala svoje Q-vrijednosti na temelju primljenih nagrada i prijelaza stanja. Broj epizoda i maksimalni koraci po epizodi su hiperparametri koji se mogu prilagoditi ovisno o računalnim resursima i ponašanju konvergencije.

|  |
| --- |
| for episode in range(num\_episodes):  total\_reward = 0  temp\_outside = np.random.randint(0, 41)  heating = np.random.randint(0, 2)  room\_temp = temp\_outside \* 0.8  state = torch.tensor([room\_temp, heating], dtype=torch.float).unsqueeze(0)    done = False  step = 0  # Track the number of steps  while not done and step < max\_steps:       action = select\_action(state, epsilon)           if action == 'heating\_on':           heating = 1           room\_temp += calculate\_added\_temperature(1, room\_temp)           if room\_temp > 35:               room\_temp = 35           room\_temp = round(room\_temp, 2)       elif action == 'heating\_off':           heating = 0           room\_temp -= 0.02       elif action == 'keep\_heating':           if heating == 1 :               room\_temp += calculate\_added\_temperature(1, room\_temp)               if room\_temp > 35:                   room\_temp = 35               room\_temp = round(room\_temp, 2)           else:               room\_temp -= 0.02         reward = get\_reward(room\_temp, heating)       total\_reward += reward  dtype=torch.float).unsqueeze(0)       next\_state = torch.tensor([room\_temp, heating], dtype=torch.float).unsqueeze(0)         done = is\_temperature\_optimal(room\_temp)         update\_q\_network(state, action, reward, next\_state, done)         state = next\_state         step += 1    if done:       print("Episode {} finished".format(episode))  else:       print("Episode {} terminated due to reaching maximum steps".format(episode)) |

Kod 4 Petlja za obuku

### Kontrolni sistem

Kontrolni sistem implementira logiku za odabir akcija na temelju naučenih Q-vrijednosti. Odlučuje hoće li uključiti grijanje, isključiti ga ili ga zadržati nepromijenjenim na temelju trenutnog stanja i naučene politike. Ovaj sistem odgovoran je za primjenu naučene politike u stvarnom vremenu za regulaciju temperature.

|  |
| --- |
| def control\_system(room\_temp, temp\_outside, heating):  global room\_temperature, temperature\_outside, heating\_on  #state = torch.tensor([room\_temp, temp\_outside, heating], dtype=torch.float).unsqueeze(0)  state = torch.tensor([room\_temp, heating], dtype=torch.float).unsqueeze(0)    with torch.no\_grad():       q\_values = q\_network(state)       action\_index = q\_values.argmax().item()  action = index\_to\_action(action\_index)  return action |

Kod 5 Kontrolni sistem

## Proces obuke

### Istraživanje-Eksploatacija

Tokom obuke, sistem uravnotežuje istraživanje i isplatu korištenjem epsilon-žedne politike. S vjerojatnošću epsilon, agent istražuje nasumičnu akciju, a s vjerojatnošću 1-epsilon, iskorištava naučenu politiku. Ovaj mehanizam osigurava da agent istražuje okolinu dovoljno kako bi naučio učinkovitu politiku dok se postupno prebacuje prema isplati kako učenje napreduje.

|  |
| --- |
| def select\_action(state, epsilon):  if np.random.rand() < epsilon:       return np.random.choice(actions)  else:       with torch.no\_grad():           q\_values = q\_network(state)           return actions[q\_values.argmax().item()] |

Kod 6 Eksploatacija

### Ažuriranje Q-Mreže

Q-mreža se ažurira koristeći Bellmanovu jednadžbu i gradijentni spust. Ciljna Q-vrijednost izračunava se na temelju primljene nagrade i maksimalne Q-vrijednosti sljedećeg stanja. Ovaj postupak ažuriranja iterativno poboljšava Q-vrijednosti prema optimalnoj politici.

|  |
| --- |
| def update\_q\_network(state, action, reward, next\_state, done):  q\_values = q\_network(state)  next\_q\_values = q\_network(next\_state)  q\_value = q\_values[0, actions.index(action)]  next\_q\_value = torch.max(next\_q\_values).item() if not done else 0  target = torch.tensor(reward + gamma \* next\_q\_value, dtype=torch.float)  # Convert target to tensor  loss = nn.MSELoss()(q\_value, target)  optimizer.zero\_grad()  loss.backward()  optimizer.step() |

Kod 7 Azuriranje

## Rezultati

### Ishod Obuke

Nakon obuke za određeni broj epizoda, sistem bi trebao naučiti učinkovitu politiku za regulaciju temperature. Naučena politika primjenjuje se u kontrolnom sistemu za dinamičko prilagođavanje sistema grijanja na temelju stvarnih očitanja temperature.

Epsilon (ε) igra ključnu ulogu u balansiranju između istraživanja i iskorištavanja tokom procesa učenja ojačavanjem. Utiče na to koliko često će agent istraživati nove akcije umjesto da se oslanja na naučenu politiku. Veće Epsilon: Kada je epsilon veći, agent će češće odabrati nasumične akcije tokom obuke. To znači da će sistem više istraživati stanja i akcije koje možda nisu bile isprobane ili su rijetke u dosadašnjem iskustvu. To može pomoći agentu da istraži različite dijelove prostora stanja i izgradi bolje razumijevanje okoline. Međutim, preveliko istraživanje može dovesti do sporijeg konvergiranja algoritma jer agent može trošiti vrijeme na akcije koje nisu optimalne.

Manje Epsilon: Kada je epsilon manji, agent će se više oslanjati na naučenu politiku tokom obuke. To znači da će sistem više iskorištavati naučene informacije i preferirati akcije koje su dosad pokazale visoku vrijednost Q-funkcije. Ovo može ubrzati konvergenciju algoritma jer se agent brže usmjerava prema optimalnim akcijama. Međutim, preveliko iskorištavanje može dovesti do suboptimalnih politika jer agent možda neće istražiti dovoljno različitih akcija i njihovih posljedica

Parametar ɣ poznat je kao faktor diskontiranja. On određuje koliko agent cijeni buduće nagrade u usporedbi s trenutnim nagradama. Promjena vrijednosti ɣ može značajno uticati na ponašanje i performanse algoritma učenja.

Visoka vrijednost ɣ(npr. 0.9 ili 0.99) čini agenta dalekovidnijim. Agent će davati prednost dugoročnim nagradama nad kratkoročnim dobicima. Agent će biti spreman poduzeti radnje koje možda neće donijeti trenutne nagrade, ali će dovesti do viših nagrada u budućnosti. To može dovesti do stabilnijeg učenja i politika koje su bolje na duge staze, ali može zahtijevati više istraživanja i duže vrijeme treniranja da se konvergira. Niska vrijednost ɣ(npr. 0.1 ili 0.3) čini agenta kratkovidnijim. Agent će davati prednost trenutnim nagradama nad budućim nagradama. Agent će preferirati radnje koje pružaju trenutne koristi, čak i ako su suboptimalne na duge staze. Proces učenja može brže konvergirati jer se agent fokusira na trenutne učinke svojih radnji, ali rezultirajuće politike mogu biti manje optimalne na duže staze. Vrijednost parametra ɣ može zahtijevati više istraživanja kako bi se naučile dugoročne koristi radnji, dok niža ɣ može brže iskoristiti poznate trenutne nagrade.

Za visoko gama 0.9 i visoko epsilon 0.5, za 1000 epoha dobili smo sljedeće rezultate.

|  |
| --- |
| Slika 1 Zavisnost ukupne nagrade od broja epizode za epsilon = 0.5 i gamma = 0.9 |
|  |

|  |
| --- |
| Slika 2 Vrijednosti parametara po epizodi za prethodnu sliku |
|  |

Vidimo da se nagrade skokovito mijenjaju što može biti uzrokovano i malim brojem epizoda(prekratak period treniranja), te veliko epsilon, jer je onda algoritam imao više istraživačkih vrijednosti za akcije i samim time su i ukupne nagrade po epizodi više varirale.

Ponovljeno je treniranje za jos 1000 epizoda sa identicnim parametrima, a onda sa tim sacuvanim podacima, nakon treninga, nastavljeno je treniranje sa promjenjenim parametrima gdje je epsilon postavljeno na 0.1 a gama zadržano na 0.9. Nakon hiljadu epizoda sa tim parametrima, odrađeno je jos 100 epizoda . Na sljedećim slikama prikazano je posljednjih 100 epizoda.

|  |
| --- |
|  |
| Slika 2.3 Zavisnost ukupne nagrade od broja epizode za epsilon = 0.1 i gamma = 0.9 |

Primjetimo da se ukupna nagrada mijenja u mnogo manjem opsegu nego u prethodnom slučaju jer su i epsilon i gama postavljeni na visoke vrijednosti, nakon što se sistemu dozvolilo treniranje u kojem je mogao poduzeti veće istraživanje i promjenu akcija.

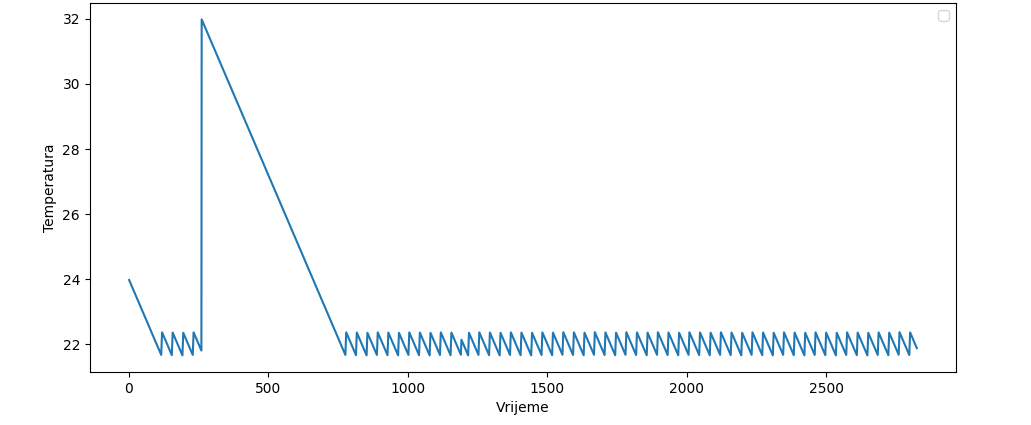
Pogledajmo sada slučaj da je epsilon malo od samog početka treninga. Neka je epsilon na početku 0.1 i neka je gama 0.9. Nakon tri 3100 epizoda imamo situaciju kao na slici 2.4. za posljednjih stotinu epizoda.

|  |
| --- |
| Slika 3 Zavisnost ukupne nagrade od broja epizode za epsilon = 0.1 i gamma = 0.9 nakon 3000 epizoda treniranih sa parametron epsilon = 0.1 |
|  |

Kao što vidimo poredeći slike 2.4 i 2.3 imamo veci broj oscilacija nagrada u zadnjih sto epizoda u slučaju da je inicijalni trening odrađen sa 0.1 vrijednosti za epsilon. Sistem koji se prvo trenira sa većim epsilon, a zatim smanji njegova vrijednost na većem broju etapa je stabilniji nego slučaj kada je epsilon konstantno malo. To je posljedica mogućnosti sistema da istražuje posljedice i nagrade slobodnije i nasumično što je epsilon veće u početnim epizodama.

|  |
| --- |
|  |
|  |
|  |

Ako su parametri zadani na način da se epsilon mijenja sa 0,5 na 0,3 a zatim na 0,1, a diskontni faktor sa 0,7 na 0,9 u 10000 epizoda imamo da se sada nas regulator ponaša kako je rikazano na slici, temperatura ce da opada sa pocetne zadane vrijednosti dok ne postigne optimalnu i onda će temperaturu zadržati do daljnjeg i sposoban je da odgovara na promjene vanjske temperature (koje naravno utiču na tempraturu unutar sobe)



Ponovo je vidljivo da nakon odredjenog vremenskog perioda temperatura ostaje u optimalnom opsegu.

### Ograničenja

RL(reinforcement learning) podložno je nekoliko izazova, uključujući problem eksplodirajuće i nestajuće vrijednosti, kao i problem stabilnosti učenja. Za rješavanje ovih izazova, važno je pravilno postaviti hiperparametre, odabrati odgovarajuću arhitekturu neuronske mreže i primijeniti tehnike kao što su iskustveno učenje i algoritmi optimizacije poput Adam-a.

## Zaključak

Sistem kontrole temperature prostorije pomoću RL predstavlja učinkovit pristup za automatsku regulaciju temperature. Kombinacija neuronskih mreža, epsilon politike i Bellmanove jednačine omogućuje agentu učenje optimalne politike za upravljanje sistemom grijanja. Kroz iterativni proces obuke, sistem postiže sposobnost održavanja temperature unutar željenog raspona čak i u promjenjivim okolnostima, čineći ga korisnim alatom za energetsku učinkovitost i udobnost u domaćinstvima i komercijalnim prostorima.

# Zadatak 1

## **Uvod**

Litijumske baterije igraju ključnu ulogu u modernim tehnologijama zbog svoje visoke energetske gustine, dugog životnog vijeka i pouzdanosti. Njihove performanse i karakteristike u velikoj mjeri zavise od kristalnog sistema materijala korištenih u elektrodama. U ovom izvještaju fokusiramo se na klasifikaciju litijumskih baterija prema tri kristalna sistema: monoklinski, ortorombični i triklinasti.

Korišteni dataset: <https://www.kaggle.com/datasets/divyansh22/crystal-system-properties-for-liion-batteries>

### **Kristalni sistemi i njihova svojstva**

**Monoklinski Sistem**

* **Struktura:** Monoklinski kristalni sistem se karakteriše pravougaonim prizma s jednim kosim uglom, što rezultira asimetričnom jedinicom ćelijom.
* **Svojstva:** Baterije s monoklinskim sistemom pružaju dobar balans između kapaciteta i troškova. One su relativno stabilne i imaju pristojan vijek trajanja. Međutim, dugotrajna upotreba može dovesti do degradacije performansi.

**Ortorombični Sistem**

* **Struktura**: Ortorombični kristalni sistem ima pravougaone prizme gdje su sve tri stranice različite dužine i svi uglovi su pravougli.
* **Svojstva**: Ovaj sistem je poznat po svojoj visokoj termičkoj i hemijskoj stabilnosti. Baterije s ortorombičnim kristalnim sistemom su sigurnije za upotrebu, imaju duži vijek trajanja i stabilne performanse. Međutim, one obično imaju nižu energetsku gustinu u poređenju sa nekim drugim sistemima.

**Triklinski Sistem**

* **Struktura**: Triklinasti kristalni sistem se karakteriše trodimenzionalnom mrežom gdje su sve tri stranice različite dužine, a svi uglovi su različiti i nijedan nije prav.
* **Svojstva**: Baterije s triklinastim sistemom pružaju jedinstvene prednosti u smislu fleksibilnosti dizajna i specifičnih aplikacija. Međutim, često imaju izazove u pogledu stabilnosti i paciteta, što može ograničiti njihovu široku primjenu.

## **Predprocesiranje podataka**

Podaci su predprocesirani na tri različita načina, a zatim je za svaki od tih načina izvršena klasifikacija korištenjem funkcije MLPClassifier sa sljedećim parametrima:

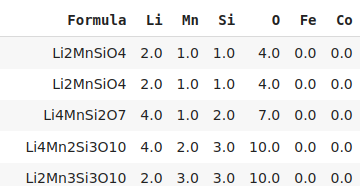
* **hidden\_layer\_sizes:** 100,
* **activation:** identity
* **solver:** sgd
* **alpha:** 0.0001
* **learning\_rate:** constant
* **learning\_rate\_init:** 0.001
* **max\_iter:** 200

**Prvi Način Predprocesiranja**:

* samo kolone sa numeričkim vrijednostima su zadržane kao karakteristike (features), dok su ostale kolone odbačene.
* Nakon toga, izvršena je klasifikacija korištenjem MLPClassifier sa gore navedenim parametrima.

**Drugi Način Predprocesiranja**:

* Od kolone pod nazivom ‘Formula’ smo napravili 6 novih kolona pod nazivima: 'Li', 'Mn', 'Si', 'O', 'Fe', 'Co', čije su vrijednosti broj atoma tog hemijskog elementa u formuli iz kolone ‘Formula’



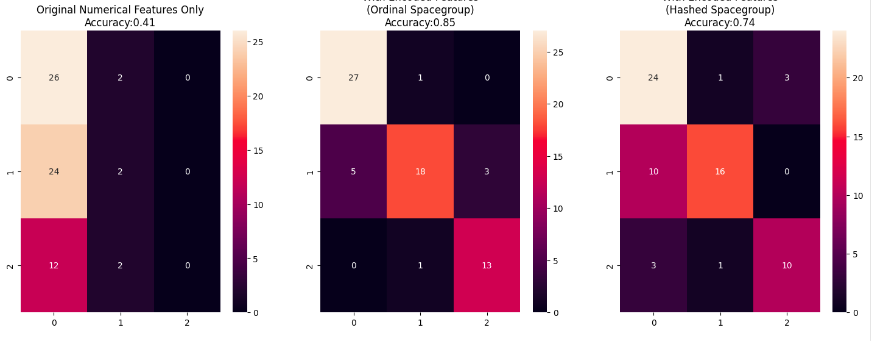
Slika 4 Kolona ‘formula’

* Svakoj unikatnoj vrijednosti iz kolone ‘Spacegroup’ smo dodijelili jedan integer
* Nakon toga, izvršena je klasifikacija korištenjem MLPClassifier sa gore navedenim parametrima.

**Treći Način Predprocesiranja**:

* U trećem načinu, transformacija kolone "Spacegroup" je izvršena drugačije u odnosu na drugi način.
* Umjesto da smo svakoj vrijednosti u koloni dodijelili jedan integer, vrijednosti u koloni su hashirane.
* Svaka hashirana vrijednost rezultira nizom od 3 elemenata
* Zatim se formiraju nove kolone (‘Spacegroup0’, ‘Spacegroup1’,’Spacegroup2’) i dodjeljuju im se vrijednosti niz[0], niz[1], niz[2] respektivno.
* Nakon toga, izvršena je klasifikacija korištenjem MLPClassifier sa podrazumijevanim parametrima.

Rezultati klasifikacije na ova tri načina su prikazani sljedećim konfuzijskim matricama:



Slika 5 Konfuzijske matrice

Zaključeno je da je za drugi način predprocesiranja postignuta najveća tačnost u klasifikaciji, koristeći MLP klasifikator sa gore navedenim parametrima. Nadalje će se u analizi koristiti podaci dobijeni korištenjem drugog načina predprocesiranja jer su rezultovali najboljom tačnošću.

Korišteni kod:

|  |
| --- |
| import chemparse  import pandas as pd    import matplotlib.pyplot as plt  import seaborn as sns  import numpy as np  from sklearn.preprocessing import LabelEncoder, MinMaxScaler  from sklearn.feature\_extraction import FeatureHasher  from sklearn.metrics import mean\_squared\_error,accuracy\_score, precision\_score,recall\_score,f1\_score,classification\_report, confusion\_matrix, ConfusionMatrixDisplay  from sklearn.tree import DecisionTreeClassifier  from sklearn.neural\_network import MLPClassifier  from sklearn.model\_selection import train\_test\_split,RandomizedSearchCV,GridSearchCV,cross\_val\_score  from sklearn.metrics import roc\_auc\_score,roc\_curve  from sklearn.base import clone  ## Pre-processsing  # Load and display data  url = 'http://apmonitor.com/pds/uploads/Main/lithium\_ion.txt'  data = pd.read\_csv(url)  num\_feat = data.select\_dtypes(include=['int64','float64']).columns  cat\_feat = data.select\_dtypes(include=['object','bool']).columns  data.drop(columns=['Materials Id'], inplace=True)    # One-hot encode 'Has Bandstructure'  data['Has Bandstructure'] = data['Has Bandstructure'].map({True: 1, False: 0})    # Feature engineering  data['Spacegroup (ordinal)'] = pd.factorize(data['Spacegroup'])[0]    n = 3  # Convert 'Spacegroup' column into iterables of strings  spacegroup\_iterables = data['Spacegroup'].apply(list)    # Use FeatureHasher  fh = FeatureHasher(n\_features=n, input\_type='string')  hashed\_tag = fh.fit\_transform(spacegroup\_iterables).toarray()    ht\_df = pd.DataFrame(hashed\_tag, columns=['Spacegroup' + str(i) for i in range(n)])  data = data.join(ht\_df)    chem\_data = data['Formula'].apply(chemparse.parse\_formula)  chem\_data = pd.json\_normalize(chem\_data).fillna(0)  data = data.join(chem\_data)    labelencoder = LabelEncoder()  data['Crystal System (#)'] = labelencoder.fit\_transform(data['Crystal System'])    data[['monoclinic', 'orthorhombic', 'triclinic']] = pd.get\_dummies(data['Crystal System'])    data.to\_csv('lithium\_ion\_data.csv', index=False)    # Model training and evaluation  # Define features and labels  features = list(data.select\_dtypes(include=['int64','float64']).columns.values)  ord\_feat = ['Formation Energy (eV)', 'E Above Hull (eV)', 'Band Gap (eV)', 'Nsites', 'Density (gm/cc)', 'Volume',  'Has Bandstructure', 'Spacegroup (ordinal)', 'Li', 'Mn', 'Si', 'O', 'Fe', 'Co']  hash\_feat = ['Formation Energy (eV)', 'E Above Hull (eV)', 'Band Gap (eV)', 'Nsites', 'Density (gm/cc)', 'Volume',  'Has Bandstructure', 'Spacegroup0', 'Spacegroup1', 'Spacegroup2', 'Li', 'Mn', 'Si', 'O', 'Fe', 'Co']  labels = ['Crystal System (#)']    # Plotting confusion matrices    best\_mlp = None  best\_mlp2 = None  best\_accuracy = 0.2    plt.figure(figsize=(18, 6))  titles = ['Original Numerical Features Only',  'With Encoded Features\n(Ordinal Spacegroup)',  'With Encoded Features\n(Hashed Spacegroup)']  for i, feat in enumerate([num\_feat, ord\_feat, hash\_feat]):  X = data[feat]  y = data[labels].values.ravel()    Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.2,random\_state=3)    mlp = MLPClassifier(activation = 'identity',solver = 'sgd',random\_state = 42)  #mlp = clone(best\_mlp2)  #mlp.activation = 'relu'  mlp.fit(Xtrain, ytrain)  yp = mlp.predict(Xtest)    # Plot confusion matrix  plt.subplot(1, 3, i + 1)  cm = confusion\_matrix(ytest, yp)  sns.heatmap(cm, annot=True)  accuracy = accuracy\_score(ytest, yp).round(2)  if(accuracy > best\_accuracy) :  best\_accuracy = accuracy  best\_mlp = mlp  best\_mlp2 = mlp  plt.title(titles[i] + "\nAccuracy:" + str(accuracy))    plt.savefig('diff\_preprocessing\_results.png') # Save the plot to a file |

Kod 8

## Podešavanje(tuning) modela a)

Podešavanje modela je eksperimentalni proces pronalaženja optimalnih vrijednosti hiperparametara kako bi se maksimizirale performanse modela. U ovom slućaju korištene su različite aktivacijske funkcije kao što su :

* relu
* Tanh
* Identity

Pored različitih aktivacijskih funkcija za podešavanje se koristio i različit broj epoha :

* 700
* 2000

Za ovaj proces korišten je najbolji model iz prethodnog koraka ,odnosno model koji je dobijen treniranjem modela korštenjem ordinalnih featura (kolona Spacegroup je ordinalno kodirana).  
Taj model je dao tačnost (accuracy) od 85% . Prije samog tuninga prvo smo morali napraviti train\_test\_split gdje je X\_ određeno sa data[ord\_feat] . Kod train\_test splita korišten je random\_state = 3 kako bi pri svakom pokretanju bloka koda dobili isti split.

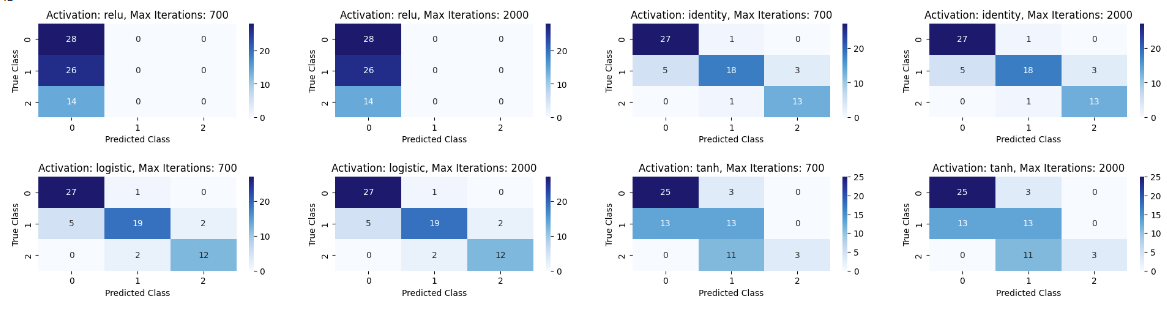
|  |
| --- |
| X\_ = data[ord\_feat]  y\_ = data[labels] .values.ravel()  #y2=data[labels]  # Train-test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_, y\_, test\_size=0.2, random\_state=3) |

Kod 9 train\_test\_split

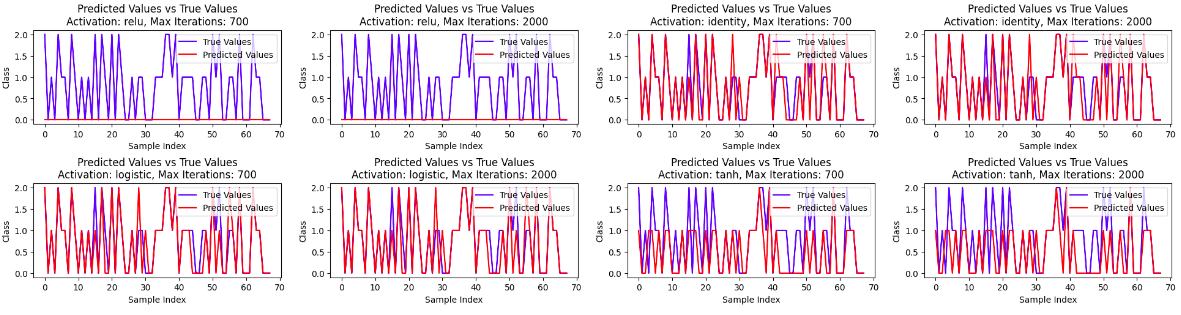
U nastavku ćemo prikazati kod za sam proces tuninga modela . Kod funkcioniše na način da se u svakoj iteraciji klonira najbolji model , te se na njemu podese novi parametri i model se ponovo trenira i evaluira . Rezultati evoluacijske metrike se prikazuju u tabeli i bar grafikonu. Prikazuju se također očekivani izlazi i konfuzijska matrica .

|  |
| --- |
| # Zadatak pod a)  # Define parameter grid  param\_grid = {  'activation': ['relu', 'identity', 'logistic', 'tanh'],  'max\_iter': [700, 2000]  }  #best\_mlp = MLPClassifier (activation ='identity',solver = 'sgd')  #best\_mlp.fit(X\_train,y\_train)  #y\_pred = best\_mlp.predict(X\_test)  #best\_accuracy = accuracy\_score(ytest, y\_pred)  print(best\_accuracy)  print(best\_mlp2.get\_params())  scores\_a = []  # Initialize the results list  results\_list = []  plt.figure(figsize=(24, 15))  # Perform manual GridSearchCV to capture all metrics  for i, activation in enumerate(param\_grid['activation']):  for j, max\_iter in enumerate(param\_grid['max\_iter']):    fold\_results = {  'activation': activation,  'max\_iter': max\_iter,  'rmse': [],  'accuracy': [],  'precision': [],  'recall': [],  'TP': [],  'TN': [],  'FP': [],  'FN': []  }  name = f'mlp{i\*len(param\_grid["max\_iter"])+j+1}'  #mlp = MLPClassifier(activation=activation, max\_iter=max\_iter,solver = 'sgd')  mlp = clone(best\_mlp)  print(mlp.random\_state)  mlp.set\_params(activation=activation, max\_iter=max\_iter)  # Train the model  mlp.fit(X\_train, y\_train)  y\_pred = mlp.predict(X\_test)  # Calculate confusion matrix  cm = confusion\_matrix(y\_test, y\_pred)  # Calculate TP, TN, FP, FN for each class  TP = np.diag(cm)  FP = np.sum(cm, axis=0) - TP  FN = np.sum(cm, axis=1) - TP  TN = np.sum(cm) - (TP + FP + FN)  # Define subplot positions  plt.subplot(5, 4, i \* len(param\_grid['max\_iter']) + j + 1)  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  plt.title(f'Activation: {activation}, Max Iterations: {max\_iter}')  plt.xlabel('Predicted Class')  plt.ylabel('True Class')  plt.savefig('MLP\_konfmatrica\_a.png')  # Calculate metrics  rmse = (np.sqrt(mean\_squared\_error(y\_test, y\_pred))).round(2)  accuracy = accuracy\_score(y\_test, y\_pred).round(2)  if accuracy > best\_accuracy :  best\_accuracy = accuracy  best\_mlp2 = mlp  best\_mlp = mlp  precision = precision\_score(y\_test, y\_pred, average='weighted').round(2)  recall = recall\_score(y\_test, y\_pred, average='weighted').round(2)  score\_dict = {'Classifier': name,  'Accuracy': accuracy,  'Precision': precision,  'Recall': recall,  'RMSE': rmse }  scores\_a.append(score\_dict)  fold\_results['rmse'].append(rmse)  fold\_results['accuracy'].append(accuracy)  fold\_results['precision'].append(precision)  fold\_results['recall'].append(recall)  fold\_results['TP'] = TP  fold\_results['TN'] = TN  fold\_results['FP'] = FP  fold\_results['FN'] = FN  plt.subplot(5, 4, i \* len(param\_grid['max\_iter']) + j + 13) # Za predikcije  plt.plot(y\_test, color='blue', label='True Values')  plt.plot(y\_pred, color='red', label='Predicted Values')  plt.title(f'Predicted Values vs True Values\nActivation: {activation}, Max Iterations: {max\_iter}')  plt.xlabel('Sample Index')  plt.ylabel('Class')  plt.legend()  results\_list.append(fold\_results)  plt.subplots\_adjust(hspace=0.63)  results\_df = pd.DataFrame(results\_list).round(2).astype(str)  plt.figure(figsize=(20, 10))  # Plot the table  plt.axis('off') # Turn off the axis  tbl = plt.table(cellText=results\_df.values, colLabels=results\_df.columns, cellLoc='center', loc='center')  tbl.auto\_set\_font\_size(False)  tbl.set\_fontsize(10)  tbl.scale(1.2, 3.6)  # Adjust layout and show plot  plt.tight\_layout()  plt.savefig('MLP\_tabela\_a.png')  model\_compare = pd.DataFrame(scores\_a)  # Plotting  model\_compare.set\_index('Classifier', inplace=True) # Postavljanje klasifikatora kao indeksa  model\_compare.plot(kind='bar', figsize=(10, 6))  plt.title('Performance Comparison of Classifiers')  plt.xlabel('Classifier')  plt.ylabel('Score')  plt.xticks(rotation=45) # Rotacija oznaka na x-osi za bolju čitljivost  plt.grid(axis='y') # Prikaz mreže na y-osi  plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left') # Prikaz legende izvan grafikona  plt.tight\_layout() # Postavljanje razmaka između podataka radi bolje čitljivosti  plt.show() |

Kod 10 Tuning a)



Slika 6 Konfuzijske matrice a)



Slika 7 Očekivani izlazi a)

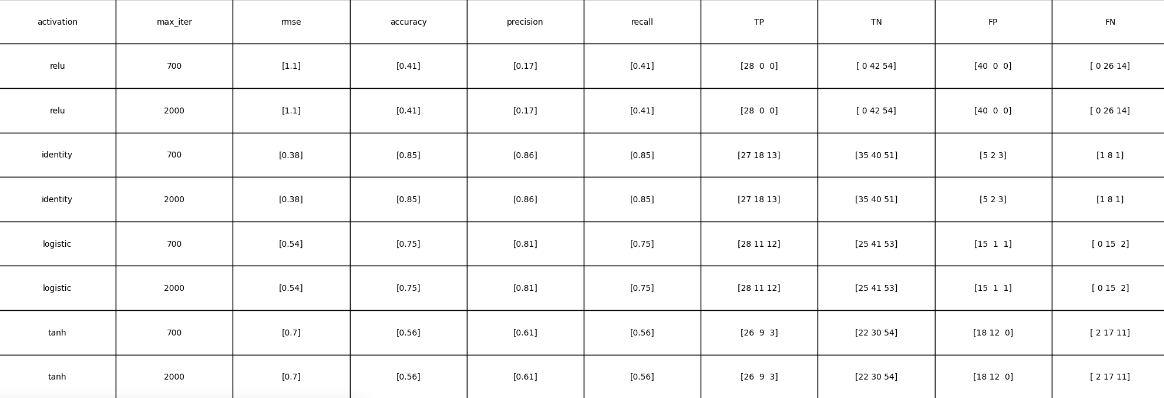
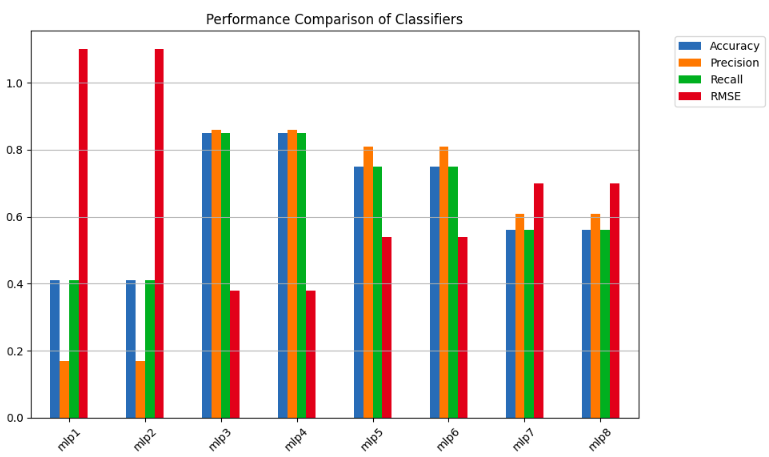


Tabela 1 Evaluacijska metrika a)

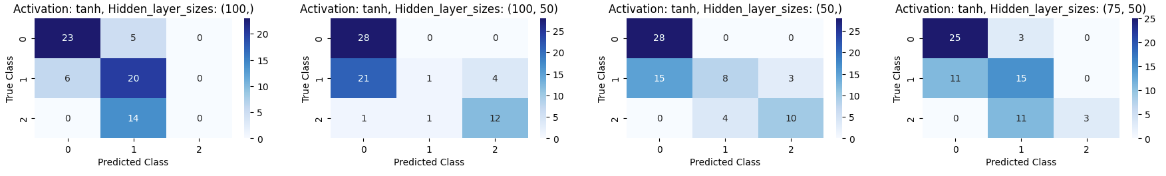


Slika 8 Usporedba klasifikatora

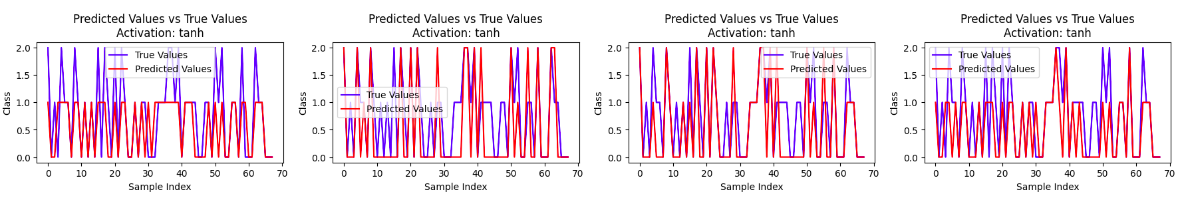
## Podešavanje (tuning) modela b)

|  |
| --- |
| # Zadatak pod b)  print(best\_accuracy)  print(best\_mlp2.get\_params())  param\_grid = {  'activation': ['tanh'],  'hidden\_layer\_sizes': [(100,), (100, 50), (50,),(75,50)]  }  # Initialize the results list  results\_list = []  plt.figure(figsize=(22, 10))    scores\_b = []  # Perform manual GridSearchCV to capture all metrics  for i, activation in enumerate(param\_grid['activation']):  for j, layers in enumerate(param\_grid['hidden\_layer\_sizes']):  fold\_results = {  'activation': activation,  'hidden\_layer\_sizes' : layers,  'rmse': [],  'accuracy': [],  'precision': [],  'recall': [],  'TP': [],  'TN': [],  'FP': [],  'FN': []  }  name = f'mlp{i\*len(param\_grid["hidden\_layer\_sizes"])+j+1}'  #mlp = MLPClassifier(max\_iter=500,activation=activation, hidden\_layer\_sizes=layers, random\_state=3)  mlp = clone(best\_mlp)  mlp.set\_params(activation=activation, hidden\_layer\_sizes=layers, random\_state=3)  # Train the model  mlp.fit(X\_train, y\_train)  y\_pred = mlp.predict(X\_test)  # Calculate confusion matrix  cm = confusion\_matrix(y\_test, y\_pred)  # Calculate TP, TN, FP, FN for each class  TP = np.diag(cm)  FP = np.sum(cm, axis=0) - TP  FN = np.sum(cm, axis=1) - TP  TN = np.sum(cm) - (TP + FP + FN)  # Define subplot positions  plt.subplot(3, 4, i \* len(param\_grid['hidden\_layer\_sizes']) + j + 1)  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  plt.title(f'Activation: {activation}, Hidden\_layer\_sizes: {layers}')  plt.xlabel('Predicted Class')  plt.ylabel('True Class')  plt.savefig('MLP\_konfmatrica\_b.png')  # Calculate metrics  rmse = (np.sqrt(mean\_squared\_error(y\_test, y\_pred))).round(2)  accuracy = accuracy\_score(y\_test, y\_pred).round(2)  if accuracy > best\_accuracy :  best\_accuracy = accuracy  best\_mlp2 = mlp  best\_mlp = mlp  precision = precision\_score(y\_test, y\_pred, average='weighted').round(2)  recall = recall\_score(y\_test, y\_pred, average='weighted').round(2)  score\_dict = {'Classifier': name,  'Accuracy': accuracy,  'Precision': precision,  'Recall': recall,  'RMSE': rmse }  scores\_b.append(score\_dict)    fold\_results['rmse'].append(rmse)  fold\_results['accuracy'].append(accuracy)  fold\_results['precision'].append(precision)  fold\_results['recall'].append(recall)  fold\_results['TP'] = TP  fold\_results['TN'] = TN  fold\_results['FP'] = FP  fold\_results['FN'] = FN    plt.subplot(3, 4, i \* len(param\_grid['hidden\_layer\_sizes']) + j + 9) # Za predikcije  plt.plot(y\_test, color='blue', label='True Values')  plt.plot(y\_pred, color='red', label='Predicted Values')  plt.title(f'Predicted Values vs True Values\nActivation: {activation}')  plt.xlabel('Sample Index')  plt.ylabel('Class')  plt.legend()  results\_list.append(fold\_results)    plt.subplots\_adjust(hspace=0.63) #razmak izmedju redova  # Create DataFrame from results\_list  results\_df = pd.DataFrame(results\_list).astype(str)  plt.figure(figsize=(20, 10))  # Plot the table  plt.axis('off') # Turn off the axis  tbl = plt.table(cellText=results\_df.values, colLabels=results\_df.columns, cellLoc='center', loc='center')  tbl.auto\_set\_font\_size(False)  tbl.set\_fontsize(10)  tbl.scale(1.2, 3.6)  # Adjust layout and show plot  plt.tight\_layout()  plt.savefig('MLP\_tabela\_b.png')  model\_compare = pd.DataFrame(scores\_b)  # Plotting  model\_compare.set\_index('Classifier', inplace=True) # Postavljanje klasifikatora kao indeksa  model\_compare.plot(kind='bar', figsize=(10, 6))  plt.title('Performance Comparison of Classifiers')  plt.xlabel('Classifier')  plt.ylabel('Score')  plt.xticks(rotation=45) # Rotacija oznaka na x-osi za bolju čitljivost  plt.grid(axis='y') # Prikaz mreže na y-osi  plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left') # Prikaz legende izvan grafikona  plt.tight\_layout() # Postavljanje razmaka između podataka radi bolje čitljivosti  plt.show() |

Kod 14 Tuning b)



Slika 9 Konfuzijske matrice



Slika 10 Očekivani izlazi

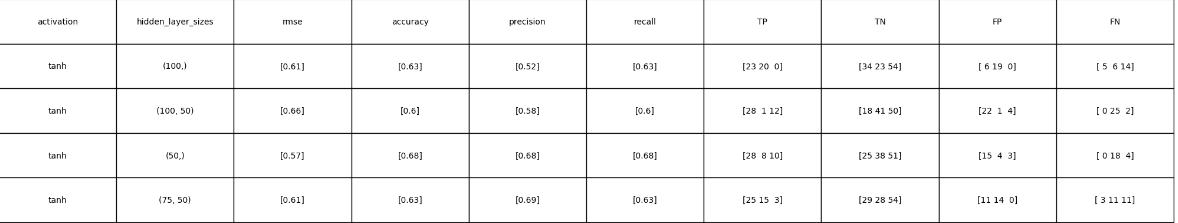
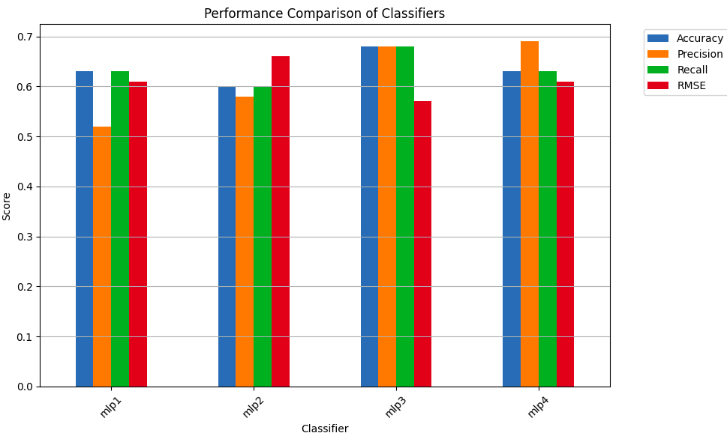


Tabela 2 rezultati



Slika 11 Performanse

## Rješenje c)

|  |
| --- |
| # Zadatak pod c)  print(best\_accuracy)  print(best\_model.get\_params())  param\_grid = {    'learning\_rate': ['constant', 'invscaling', 'adaptive'], # Dodajte različite vrijednosti za learning rate  'learning\_rate\_init': [0.001, 0.01, 0.1] # Dodajte različite vrijednosti learning ratea  }    # Initialize the results list  results\_list = []  plt.figure(figsize=(30, 20))  scores\_c = []    # Perform manual GridSearchCV to capture all metrics  for i, learning\_rate in enumerate(param\_grid['learning\_rate']):  for j, learning\_init in enumerate(param\_grid['learning\_rate\_init']):    fold\_results = {  'learning\_rate': learning\_rate,  'learning\_rate\_init': learning\_init,  'rmse': [],  'accuracy': [],  'precision': [],  'recall': [],  'TP': [],  'TN': [],  'FP': [],  'FN': []  }  name = f'mlp{i\*len(param\_grid["learning\_rate\_init"])+j+1}'    #mlp = MLPClassifier(max\_iter=500,learning\_rate\_init=learning\_init, learning\_rate=learning\_rate, random\_state=3)  mlp = clone(best\_model)  mlp.set\_params(max\_iter=500,learning\_rate\_init=learning\_init, learning\_rate=learning\_rate, random\_state=3)  # Train the model  mlp.fit(X\_train, y\_train)  y\_pred = mlp.predict(X\_test)    # Calculate confusion matrix  cm = confusion\_matrix(y\_test, y\_pred)  # Calculate TP, TN, FP, FN for each class  TP = np.diag(cm)  FP = np.sum(cm, axis=0) - TP  FN = np.sum(cm, axis=1) - TP  TN = np.sum(cm) - (TP + FP + FN)    # Define subplot positions  plt.subplot(7, 3, i \* len(param\_grid['learning\_rate\_init']) + j + 1)    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  plt.title(f'Learning rate: {learning\_rate}, \nLearning\_rate\_init: {learning\_init}')  plt.xlabel('Predicted Class')  plt.ylabel('True Class')  plt.savefig('MLP\_konfmatrica\_c.png')    # Calculate metrics  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  accuracy = accuracy\_score(y\_test, y\_pred)  if accuracy > best\_accuracy :  best\_accuracy = accuracy  best\_model = mlp  precision = precision\_score(y\_test, y\_pred, average='weighted', zero\_division=0)  recall = recall\_score(y\_test, y\_pred, average='weighted', zero\_division=0)    score\_dict = {'Classifier': name,  'Accuracy': accuracy,  'Precision': precision,  'Recall': recall,  'RMSE': rmse }  scores\_c.append(score\_dict)  fold\_results['rmse'].append(rmse)  fold\_results['accuracy'].append(accuracy)  fold\_results['precision'].append(precision)  fold\_results['recall'].append(recall)  fold\_results['TP'] = TP  fold\_results['TN'] = TN  fold\_results['FP'] = FP  fold\_results['FN'] = FN    plt.subplot(7, 3, i \* len(param\_grid['learning\_rate\_init']) + j + 13) # Za predikcije  plt.plot(y\_test, color='blue', label='True Values')  plt.plot(y\_pred, color='red', label='Predicted Values')  plt.title(f'Predicted Values vs True Values\nlearning\_rate: {learning\_rate}, learning\_rate\_init: {learning\_init}')  plt.xlabel('Sample Index')  plt.ylabel('Class')  plt.legend()    results\_list.append(fold\_results)  # Create DataFrame from results\_list  results\_df = pd.DataFrame(results\_list).astype(str)  #print(results\_df)  plt.figure(figsize=(20, 10))  # Plot the table  plt.axis('off') # Turn off the axis  tbl = plt.table(cellText=results\_df.values, colLabels=results\_df.columns, cellLoc='center', loc='center')  tbl.auto\_set\_font\_size(False)  tbl.set\_fontsize(10)  tbl.scale(1.2, 3.6)    # Adjust layout and show plot  plt.tight\_layout()  plt.savefig('MLP\_tabela\_c.png')    # Plotting  model\_compare = pd.DataFrame(scores\_c)    model\_compare.set\_index('Classifier', inplace=True) # Postavljanje klasifikatora kao indeksa  model\_compare.plot(kind='bar', figsize=(10, 6))  plt.title('Performance Comparison of Classifiers')  plt.xlabel('Classifier')  plt.ylabel('Score')  plt.xticks(rotation=45) # Rotacija oznaka na x-osi za bolju čitljivost  plt.grid(axis='y') # Prikaz mreže na y-osi  plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left') # Prikaz legende izvan grafikona  plt.tight\_layout() # Postavljanje razmaka između podataka radi bolje čitljivosti  plt.show() |

Kod 15

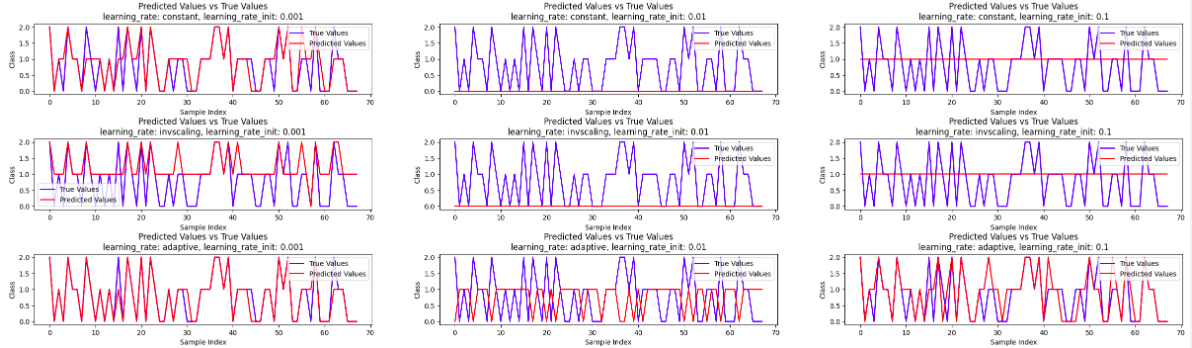


Tabela 3 Očekivani izlazi

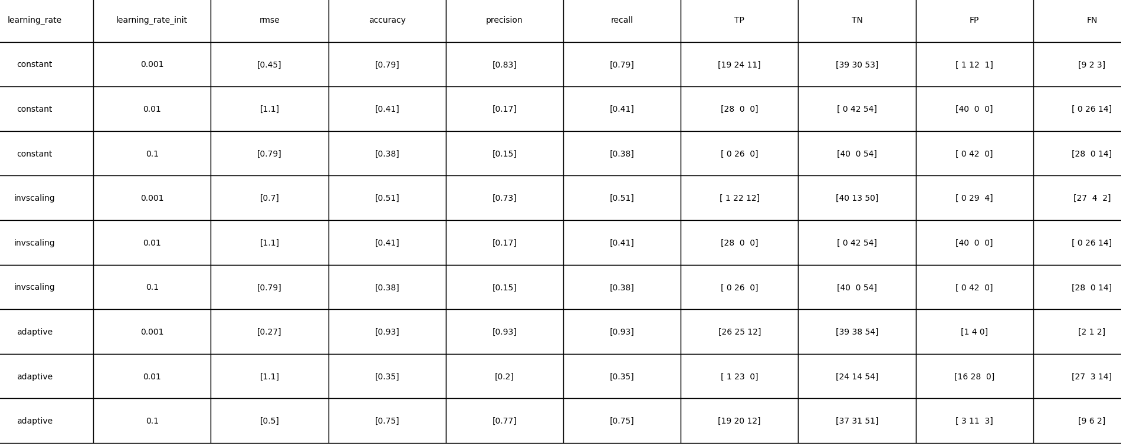
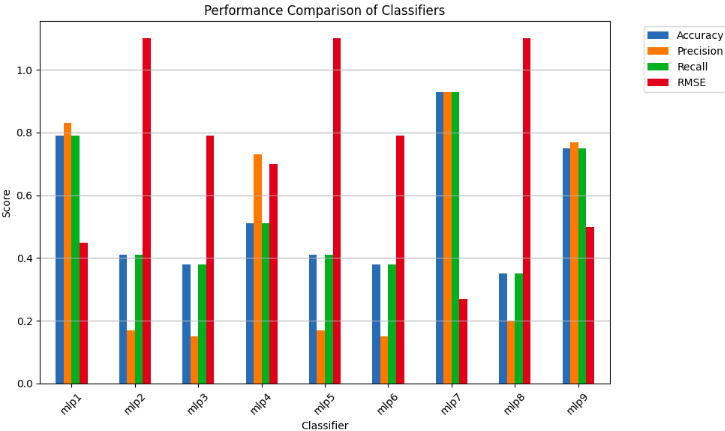
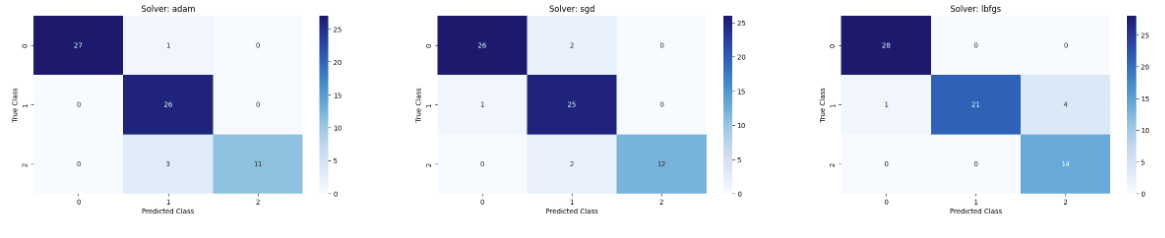


Tabela 4 rezultati c)

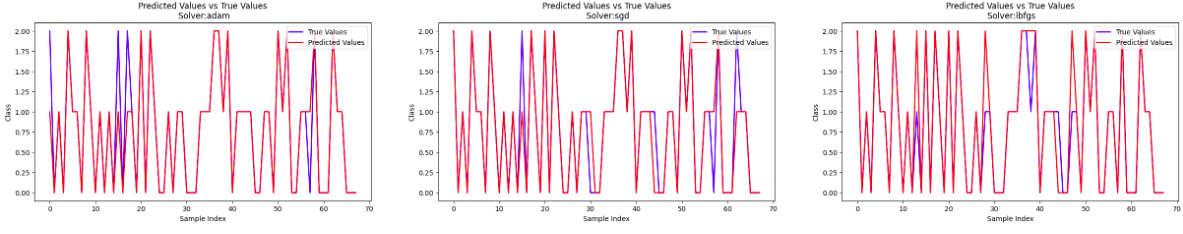


Slika 12 Usporedba rezultata

## Rješenje d)



Slika 13 Konfuzijske matrice



Slika 14 Ocekivani izlazi

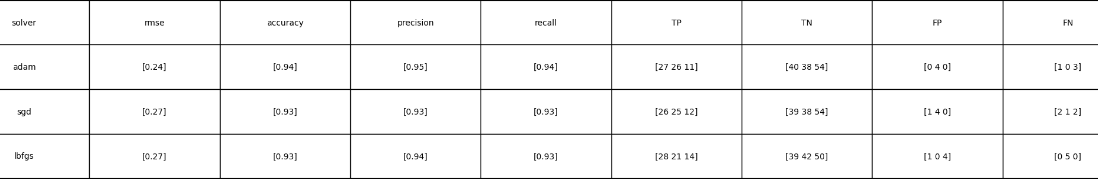
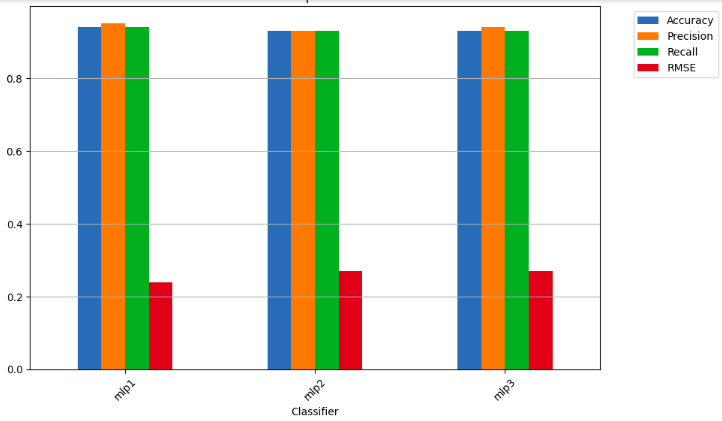


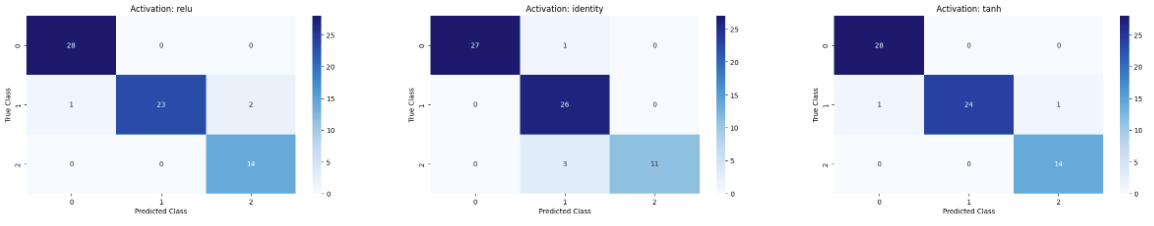
Tabela 5 rjesenja d)



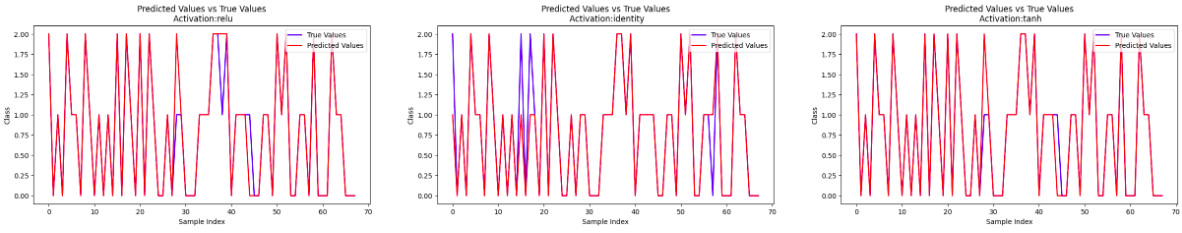
Slika 15 Usproedba d)

## Dodatno podesavanje e)

Izvršeno je dodatno podešavanje pomoću kojeg su dobijene najbolje vrijednosti tačnosti, preciznosti, odziva i srednje kvadratne greške. Tačnost modela podešenog u ovom koraku iznosi 97% i on predstavlja naš zlatni model. Korišten je najbolji model iz prethodnog koraka pod d) koji je davao tačnost od 94%. Zatim smo na njemu primjenili različite aktivacijske funkcije i dobili bollja rješenja.



Slika 16 Konfuzijske matrice e)



Slika 17 Očekivani izlazi

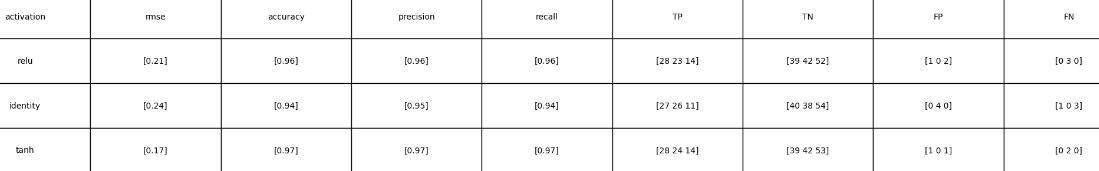
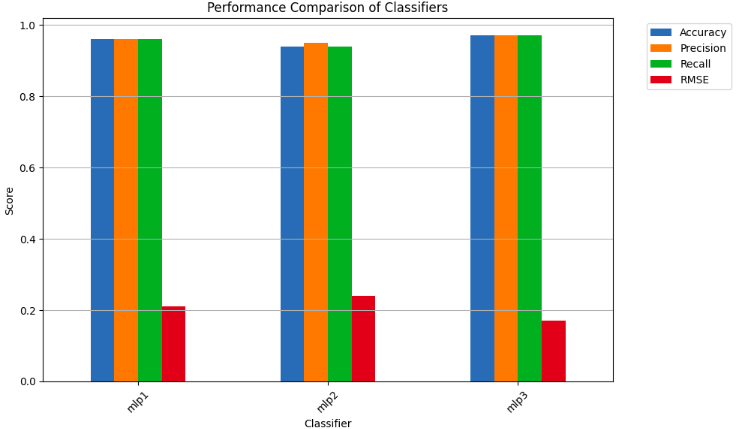


Tabela 6 Rezultati



Slika 18 Usporedba klasifikatora