Univerzitet u Kragujevcu

Fakultet inženjerskih nauka



**Veštačka inteligencija**

Projektni zadatak:

Teaching Assistant Evaluation

Student: Predmetni nastavnik:

Nikola Ivanović 637/2019 Prof. Vesna Ranković

Sadržaj

[1. Opis zadatog problema 3](#_Toc136111523)

[2. Opis ulaznih i izlaznih podataka 3](#_Toc136111524)

[3. Opis programa 4](#_Toc136111525)

[4. Dodatna rešenja sa različitim algoritmima 17](#_Toc136111526)

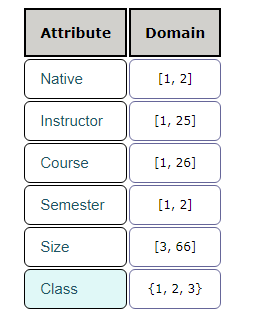
[5. Literatura 22](#_Toc136111527)

# Opis zadatog problema

Projekat se odnosi na analizu podataka o ocenama nastavnih performansi za 151 asistentsku poziciju (TA) u Odeljenju za statistiku na Univerzitetu u Wisconsin-Madisonu. Podaci obuhvataju evaluacije tokom tri redovna semestra i dva letnja semestra. Ocene su podeljene u tri kategorije (niska (1), srednja (2) i visoka (3)) kako bi se formirala klasifikaciona varijabla.Uz pomoć neuralne mreže izvršićemo klasifikaciju na ove tri kategorije.

# Opis ulaznih i izlaznih podataka

Skup podataka koji se koristi za treniranje sastoji se od 6 atributa i 151 (profesora) uzoraka opisanih tim atributima. Atributi i moguće vrednosti koje se koriste u skupu podataka su sledeći (slika 1)



Slika 1) Nazivi I vrednosti atributa

Značenje navedenih atributa:

1. Rodni jezik -> da li je TA maternji govornik engleskog jezika: 1 = Maternji govornik engleskog jezika, 2 = Ne-maternji govornik engleskog jezika.

2. Instruktor -> nastavnik kursa (25 kategorija).

3. Kurs -> 26 kategorija.

4. Semestar -> letnji ili redovan semestar: 1 = Letnji, 2 = Redovan.

5. Veličina -> broj studenata u klasi (numerička vrednost).

Kao izlaznu vrednost treba dobiti tri klase (niska (1), srednja (2) i visoka (3)). Ove tri klase predstavljaju ocene koje se dobijaju prema atributima koje smo videli iznad.

# Opis programa

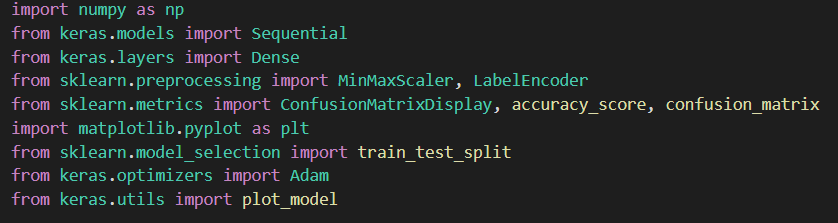
Prilikom izrade projekta korišćen je programski jezik Python zbog dostupnih biblioteka i pogodnosti koje pruža za obučavanje mreža i mašinsko učenje. Koristili smo vise vrsta algoritama (Neural Network, Perceptron, Logistic Regression, Decision Tree). Ali najbolji rezultat nam je dala Neural Network i nju smo izabrali kao glavno resenje. Sada ćemo objasniti delove koda Neural Network.

Podelicemo kod na sekcije i zatim ih objasniti pojedinačno:

1. Uvoz neophodnih biblioteka i modula.
2. Inicijalizacija skaler objekta za normalizaciju podataka, učitavanje podataka iz datoteke 'tae.dat' i njihova predobrada.
3. Normalizacija ulaznih karakteristika.
4. Podela skupa podataka na trening i test skupove.
5. Definisanje arhitekture modela.
6. Postavljanje stope učenja za Adam optimizator, kompajliranje modela sa odabranim optimizatorom i gubitkom.
7. Treniranje modela na trening skupu podataka.
8. Prikaz tačnosti modela tokom treniranja.
9. Predviđanje klasa za normalizovane podatke iz testnog skupa, Izračunavanje tačnosti modela i prikaz matrice konfuzije.
10. Prikaz matrice konfuzije.
11. Prikaz arhitekture modela u obliku grafa.

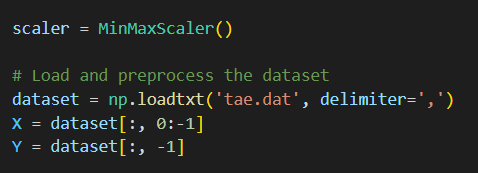
1. Uvoz neophodnih biblioteka i modula:

* `numpy` je biblioteka za rad sa numeričkim podacima u Pythonu.
* `keras.models` sadrži klasu `Sequential`, koja se koristi za kreiranje i treniranje neuronskih mreža u Kerasu.
* `keras.layers` sadrži različite slojeve koji se mogu dodati u neuronsku mrežu.
* `sklearn.preprocessing` sadrži različite alate za pretprocesiranje podataka, uključujući `MinMaxScaler`, koji se koristi za normalizaciju podataka.
* `sklearn.metrics` sadrži razne metrike koje se koriste za evaluaciju modela, kao što je `accuracy\_score` za računanje tačnosti i `confusion\_matrix` za računanje matrice konfuzije.
* `matplotlib.pyplot` se koristi za vizualizaciju podataka i rezultata.
* `sklearn.model\_selection` sadrži alate za rad sa skupovima podataka, uključujući `train\_test\_split` za podelu podataka na trening i test skupove.
* `keras.optimizers` sadrži različite optimizatore koji se mogu koristiti prilikom treniranja modela, a u ovom kodu se koristi `Adam`.
* `keras.utils` sadrži pomoćne funkcije za rad sa Kerasom, kao što je `plot\_model` koja se koristi za vizualizaciju arhitekture modela.



Slika 2) Python biblioteke

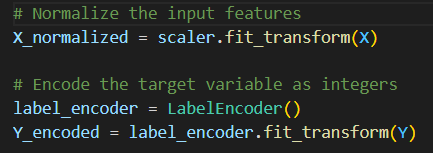
2. Inicijalizacija skaler objekta za normalizaciju podataka, učitavanje podataka iz datoteke 'tae.dat' i njihova predobrada:



Slika 3) Učitavanje podataka, njihova obrada

Ovde je MinMaxScaler inicijalizovan kako bi skalirao ulazne karakteristike između 0 i 1. Podaci se učitavaju iz datoteke 'tae.dat', gde se ulazne karakteristike čuvaju u promenljivoj X, a ciljna promenljiva u promenljivoj Y.

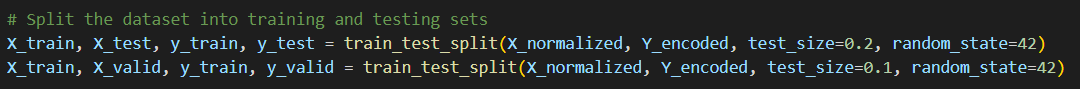
3. Normalizacija ulaznih karakteristika:



Slika 4) Normalizacija

Ulazne karakteristike X se normalizuju pomoću MinMaxScaler-a kako bi se osiguralo da su sve karakteristike na sličnoj skali. Ciljna promenljiva Y se enkodira kao celobrojne vrednosti koristeći LabelEncoder.

4. Podela skupa podataka na trening i test skupove:



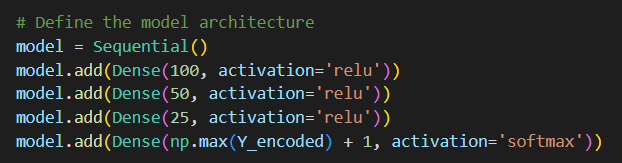
Slika 5) Podela podataka

Prvi red koda deli normalizovane ulazne karakteristike `X\_normalized` i kodirane ciljeve `Y\_encoded` na trening (X\_train, y\_train) i test (X\_test, y\_test) skupove. Parametar `test\_size` je postavljen na 0.2, što znači da će 20% podataka biti rezervisano za testiranje, dok će preostalih 80% biti korišćeno za trening. Parametar `random\_state` je postavljen na 42 kako bi se osigurala reprodukcija istih rezultata pri deljenju podataka.

Drugi red koda dodatno deli normalizovane ulazne karakteristike `X\_normalized` i kodirane ciljeve `Y\_encoded` na trening (X\_train, y\_train) i validacioni (X\_valid, y\_valid) skup. Parametar `test\_size` je postavljen na 0.1, što znači da će 10% podataka biti rezervisano za validaciju, dok će preostalih 90% biti korišćeno za trening. Parametar `random\_state` je ponovo postavljen na 42 kako bi se obezbedila doslednost u deljenju podataka.

Na kraju ovog koraka, sada imate četiri skupa podataka: X\_train, y\_train za trening modela, X\_test, y\_test za evaluaciju modela na neviđenim podacima tokom testiranja, i X\_valid, y\_valid za praćenje performansi modela tokom treninga pomoću validacije.

5. Definisanje arhitekture modela:



Slika 6) Arhitektura modela

U ovom kodu se koristi Sequential API iz Keras biblioteke za definisanje arhitekture modela. Sequential API omogućava jednostavno dodavanje slojeva jedan za drugim.

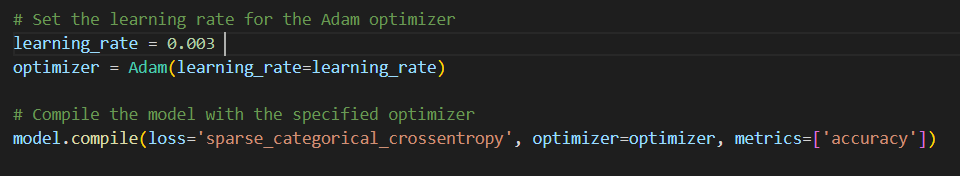
U primeru arhitekture modela, koriste se potpuno povezani (Dense) slojevi sa ReLU (Rectified Linear Unit) aktivacijom. Svaki sloj ima određeni broj neurona, koji je specificiran kao prvi argument u Dense funkciji. Aktivacija 'relu' se primenjuje na svaki neurona u tim slojevima.

Prvi sloj ima 100 neurona, drugi sloj ima 50 neurona, a treći sloj ima 25 neurona. Ovi slojevi formiraju skrivene slojeve modela, koji služe za izvlačenje složenih karateristika iz ulaznih podataka.

Poslednji sloj je izlazni sloj modela. Broj neurona u izlaznom sloju je određen kao maksimalna vrednost ciljne promenljive (Y\_encoded) plus jedan. Aktivaciona funkcija 'softmax' se koristi na izlaznom sloju kako bi se dobio vektor verovatnoća za svaku klasu. Ova funkcija se često koristi u višeklasnom klasifikaciji kako bi se normalizovali izlazi i dobile verovatnoće za svaku klasu.

Ukratko, ovaj model ima tri skrivena sloja sa ReLU aktivacijom i jedan izlazni sloj sa softmax aktivacijom za višeklasnu klasifikaciju.

6. Postavljanje stope učenja za Adam optimizator, kompajliranje modela sa odabranim optimizatorom i gubitkom:



Slika 7) Stopa učenja i kompajliranje

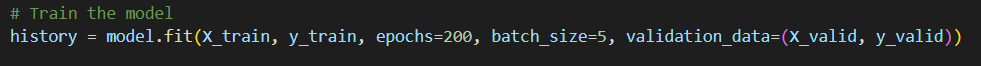
Stopa učenja (learning rate) je važan hiperparametar koji određuje koliko brzo će se model prilagođavati tokom procesa optimizacije. Za Adam optimizator, stopa učenja predstavlja korak veličine kojim će se ažurirati težine modela u svakoj iteraciji.

Kada se model kompajlira, specificira se funkcija gubitka (loss function) koja se koristi za procenu razlike između stvarnih vrednosti i predviđenih vrednosti modela. Cilj je minimizirati ovu funkciju gubitka kako bi se model prilagodio što bolje.

Takođe, optimizator se odabire za optimizaciju modela. U ovom slučaju, koristi se Adam optimizator, koji kombinuje prednosti adaptivnog momentuma (adaptive momentum) i RMSprop optimizatora. Adam optimizator prilagođava stopu učenja za svaki parametar zasebno, što mu omogućava da se prilagodi brzinama konvergencije različitih parametara.

Kada je model kompajliran, spreman je za treniranje, gde će se težine i parametri modela ažurirati iterativno kako bi se minimizovala funkcija gubitka na trening podacima.

7. Treniranje modela na trening skupu podataka:



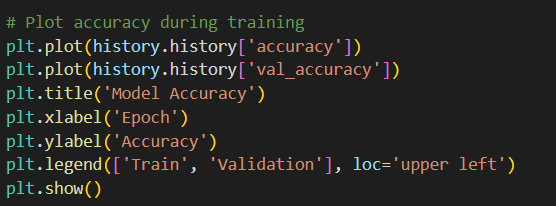
Slika 8) Treniranje modela

Model se trenira na trening podacima tokom određenog broja epoha (epochs) i veličine paketa (batch size). Epoha predstavlja prolazak kroz ceo trening skup podataka, dok veličina paketa određuje koliko uzoraka se koristi u svakoj iteraciji optimizacije.

Tokom treninga, koristi se validacioni skup podataka kako bi se evaluirala performansa modela. Na kraju svake epohe, model se evaluira na validacionom skupu kako bi se proverilo kako se performansa modela poboljšava tokom treninga.

Ovaj proces se ponavlja kroz zadati broj epoha, gde model prilagođava svoje težine i parametre kako bi minimizirao funkciju gubitka na trening skupu podataka. Cilj je da se model generalizuje i postigne dobre rezultate ne samo na trening podacima već i na novim, neviđenim podacima.

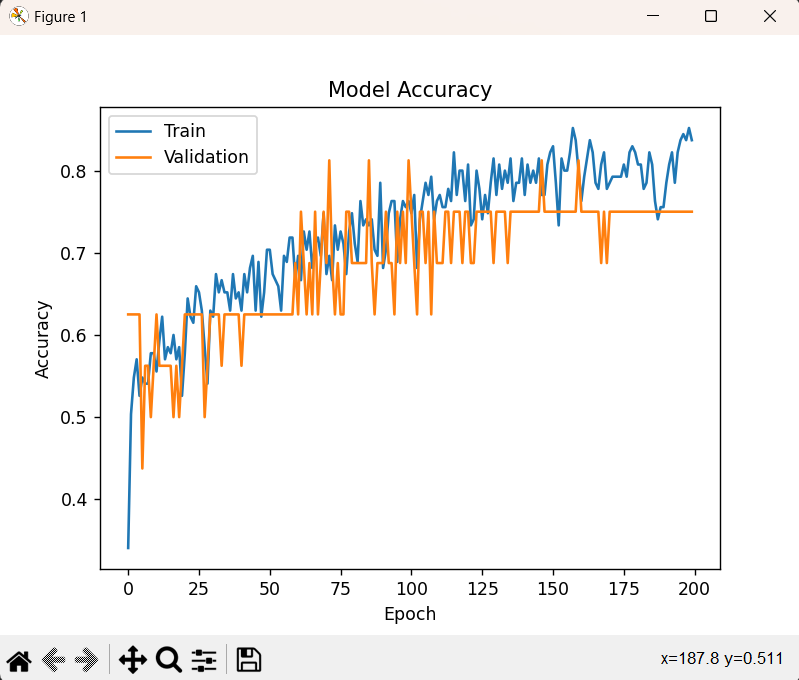
8. Prikaz tačnosti modela tokom treniranja:



Slika 9) Prikaz tačnosti

Ovaj kod se koristi za vizualizaciju tačnosti (accuracy) modela tokom treninga i validacije.

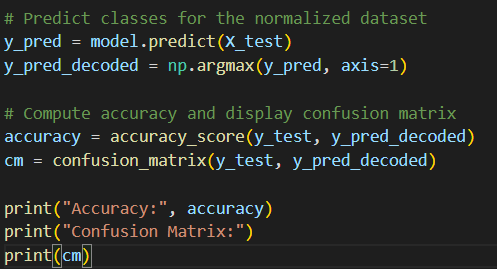
* plt.plot(history.history['accuracy']) crta liniju koja prikazuje tačnost (accuracy) modela na trening skupu tokom svake epohe.
* plt.plot(history.history['val\_accuracy']) crta liniju koja prikazuje tačnost modela na validacionom skupu tokom svake epohe.
* plt.title('Model Accuracy') postavlja naslov dijagrama na "Model Accuracy".
* plt.xlabel('Epoch') postavlja oznaku x-ose na "Epoch" (epoha), predstavljajući broj epohe.
* plt.ylabel('Accuracy') postavlja oznaku y-ose na "Accuracy" (tačnost), predstavljajući vrednosti tačnosti modela.
* plt.legend(['Train', 'Validation'], loc='upper left') postavlja legendu dijagrama, gde se "Train" odnosi na trening tačnost, a "Validation" na tačnost na validacionom skupu. "loc='upper left'" postavlja legendu u gornji levi ugao dijagrama.
* plt.show() prikazuje dijagram tačnosti modela.



Slika 10) Dijagram tačnosti

Ovim dijagramom možete pratiti kako se tačnost modela menja tokom epoha i uporediti tačnost modela na trening i validacionom skupu. Ako su linije blizu ili paralelne, to bi moglo ukazivati na dobru generalizaciju modela. U suprotnom, razlika između linija bi mogla ukazivati na overfitting ili underfitting modela.

9. Predviđanje klasa za normalizovane podatke iz testnog skupa, Izračunavanje tačnosti modela i prikaz matrice konfuzije:

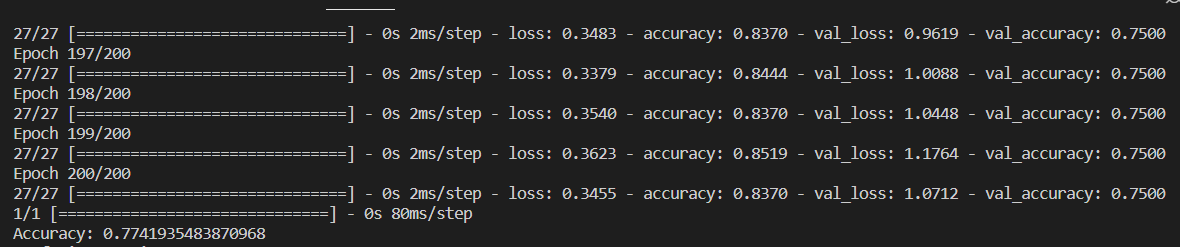


Slika 11) Predvidjanje klase, izračunavanje tačnosti i prikaz matrice konfuzije

Ovde se vrši predviđanje klasa za normalizovani skup podataka X\_test koristeći obučeni model. Nakon toga, izračunava se tačnost (accuracy) predikcija i prikazuje matrica konfuzije (confusion matrix).

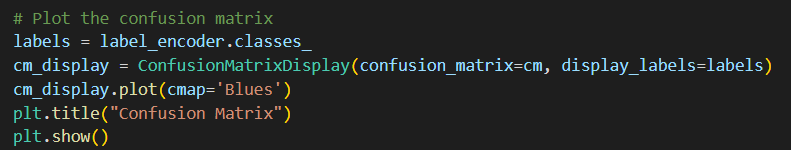
* y\_pred = model.predict(X\_test) koristi se trenirani model da bi se predvidele klase za ulazne podatke X\_test. Rezultat je vektor verovatnoća za svaku klasu za svaki podatak u X\_test.
* y\_pred\_decoded = np.argmax(y\_pred, axis=1) koristi se np.argmax funkcija kako bi se pronašao indeks klase sa najvećom verovatnoćom za svaki podatak u y\_pred. To rezultira vektorom koji sadrži predviđene klase za X\_test.
* accuracy = accuracy\_score(y\_test, y\_pred\_decoded) koristi se funkcija accuracy\_score iz scikit-learn biblioteke za izračunavanje tačnosti (accuracy) predikcija. Ova funkcija poredi prave vrednosti ciljne promenljive y\_test sa predviđenim vrednostima y\_pred\_decoded i izračunava tačnost kao udeo tačno predviđenih instanci u ukupnom broju instanci.
* cm = confusion\_matrix(y\_test, y\_pred\_decoded) koristi se funkcija confusion\_matrix iz scikit-learn biblioteke za izračunavanje matrice konfuzije (confusion matrix). Ova matrica pruža uvid u tačnost modela za svaku klasu, kao i međusobne konfuzije između klasa.
* print("Accuracy:", accuracy) prikazuje izračunatu tačnost modela.
* print("Confusion Matrix:") prikazuje poruku "Confusion Matrix:" kako bi se naglasilo da će biti prikazana matrica konfuzije.
* print(cm) prikazuje matricu konfuzije.

Ovim kodom dobijate informaciju o tačnosti modela na testiranom skupu podataka i prikazujete matricu konfuzije koja pruža detaljniji uvid u performanse modela za svaku klasu.



Slika 12) Prikaz tacnosti

10. Prikaz matrice konfuzije:

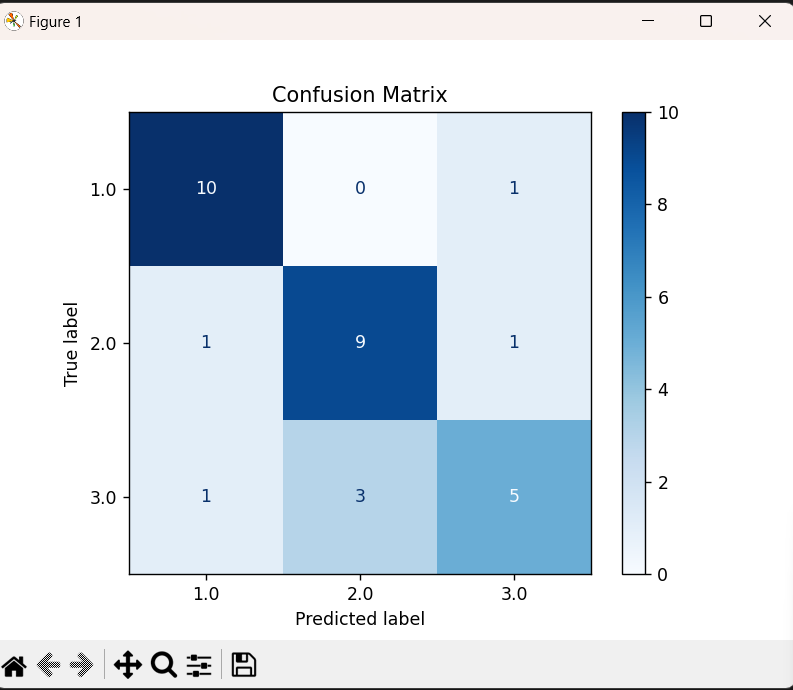


Slika 13) Grafik matrice konfuzije kod

U ovom delu koda se prikazuje matrica konfuzije (confusion matrix) koristeći funkcionalnost `ConfusionMatrixDisplay` iz biblioteke scikit-learn. Nakon što se kreira objekat `cm\_display` sa matricom konfuzije i prikaznim oznakama klasa, poziva se metoda `plot` kako bi se prikazala matrica konfuzije.

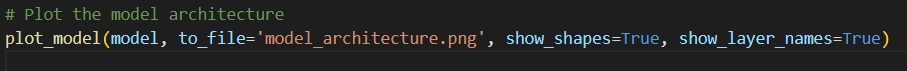
* `labels = label\_encoder.classes\_` dobija se niz oznaka klasa iz `label\_encoder`, koji su korišćeni za enkodiranje ciljnih promenljivih. Ove oznake će se koristiti za prikaz na osama matrice konfuzije.
* `cm\_display = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=labels)` kreira se objekat `cm\_display` sa matricom konfuzije `cm` i prikaznim oznakama klasa `labels`.
* `cm\_display.plot(cmap='Blues')` poziva se metoda `plot` na objektu `cm\_display` kako bi se prikazala matrica konfuzije. Parametar `cmap='Blues'` se koristi za odabir boje za prikazivanje matrice konfuzije, u ovom slučaju koristi se plava boja.
* `plt.title("Confusion Matrix")` postavlja naslov dijagrama na "Confusion Matrix".
* `plt.show()` prikazuje dijagram matrice konfuzije.

Ovaj dijagram matrice konfuzije prikazuje vizuelnu reprezentaciju performansi modela za svaku klasu. Pomaže u razumevanju koliko dobro model klasifikuje instance za svaku klasu, kao i međusobne konfuzije između klasa.



Slika 14) Grafik matrice konfuzije

11. Prikaz arhitekture modela u obliku grafa:

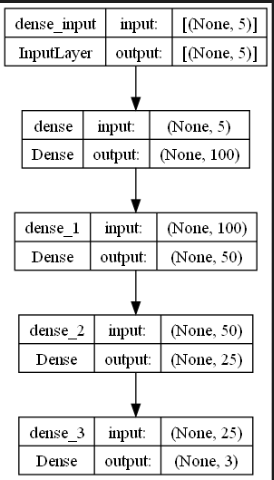


Slika 15) Izgled arhitekture modela kod

U ovom delu koda se koristi funkcija `plot\_model` iz biblioteke Keras za prikaz arhitekture modela

* Parametar `model` je objekat modela koji želimo da prikažemo.
* Parametar `to\_file='model\_architecture.png'` specificira ime datoteke u koju će se sačuvati generisani dijagram. U ovom slučaju, dijagram će biti sačuvan u datoteci "model\_architecture.png".
* Parametar `show\_shapes=True` određuje da li će se prikazati oblik ulaznih i izlaznih podataka za svaki sloj modela.
* Parametar `show\_layer\_names=True` određuje da li će se prikazati nazivi slojeva modela.

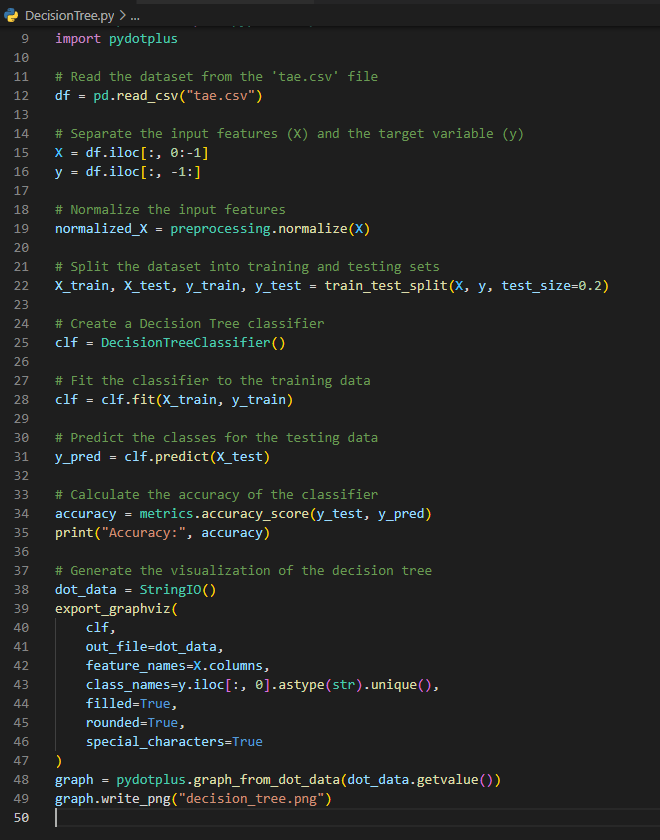
Pozivanje ove funkcije generisaće dijagram arhitekture modela i sačuvati ga u datoteku "model\_architecture.png". Dijagram prikazuje kako su slojevi povezani u modelu i omogućava vizuelni pregled strukture modela.



Slika 16) Izgled arhitekture modela

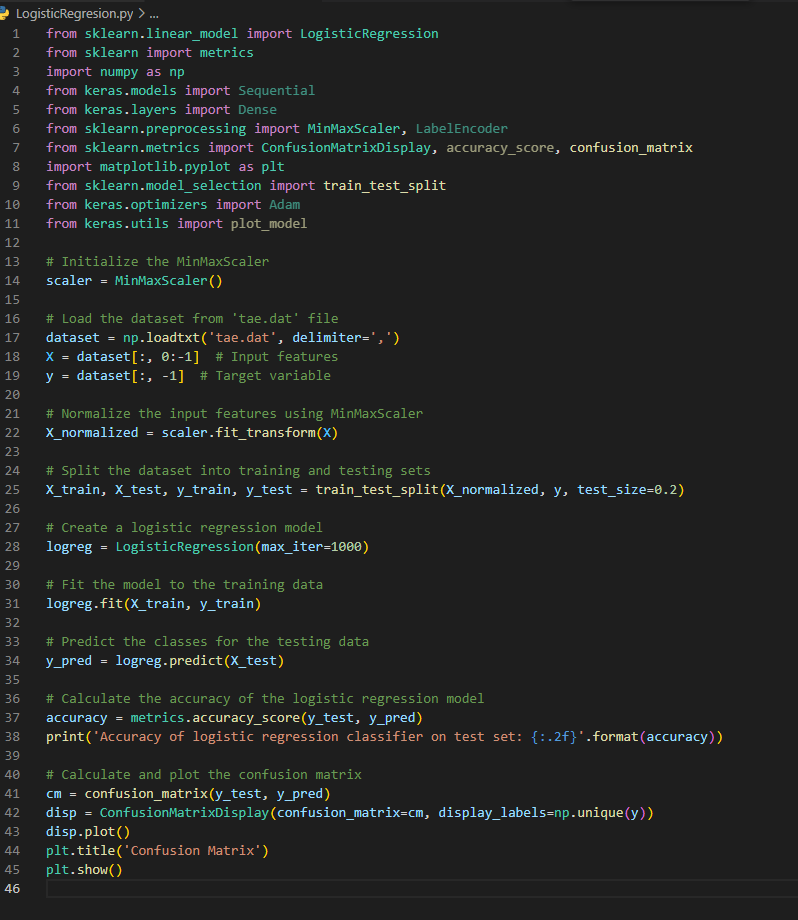
# Dodatna rešenja sa različitim algoritmima

* Decision Tree

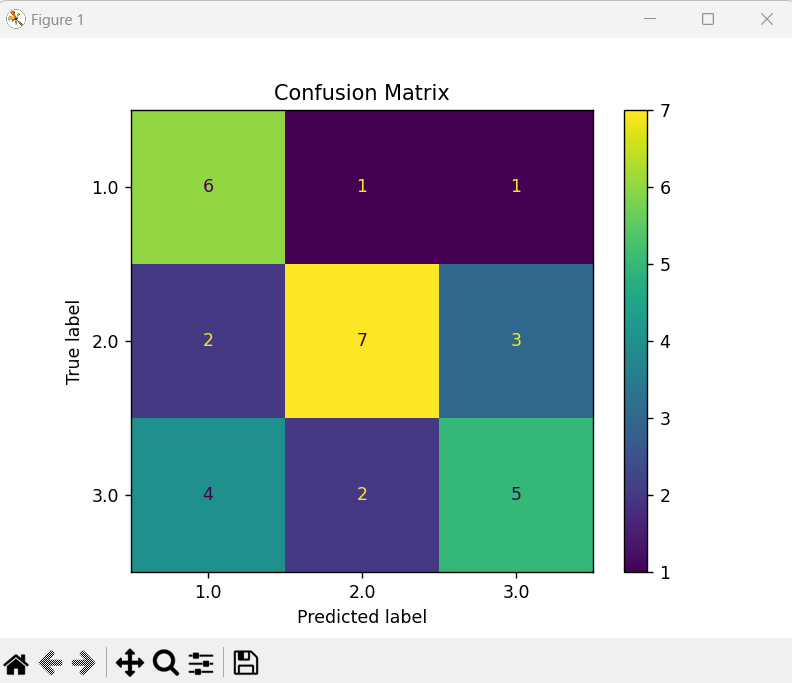


Slika 17) Kod I Accuracy od modela Decision Tree

* Logistic Regression

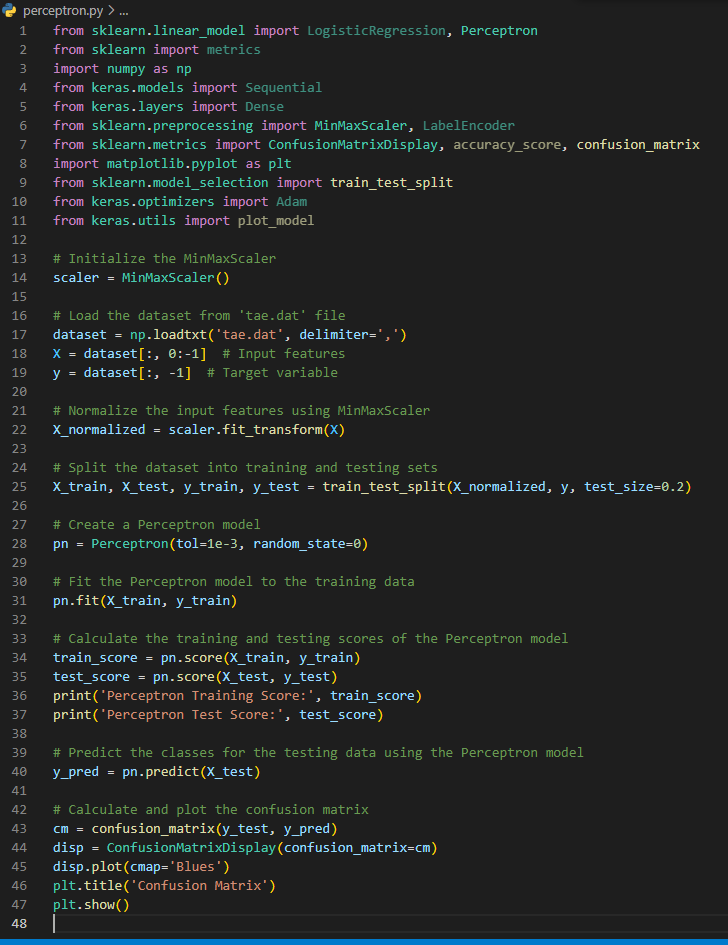


Slika 18) Kod I Accuracy od modela Logistic Regression

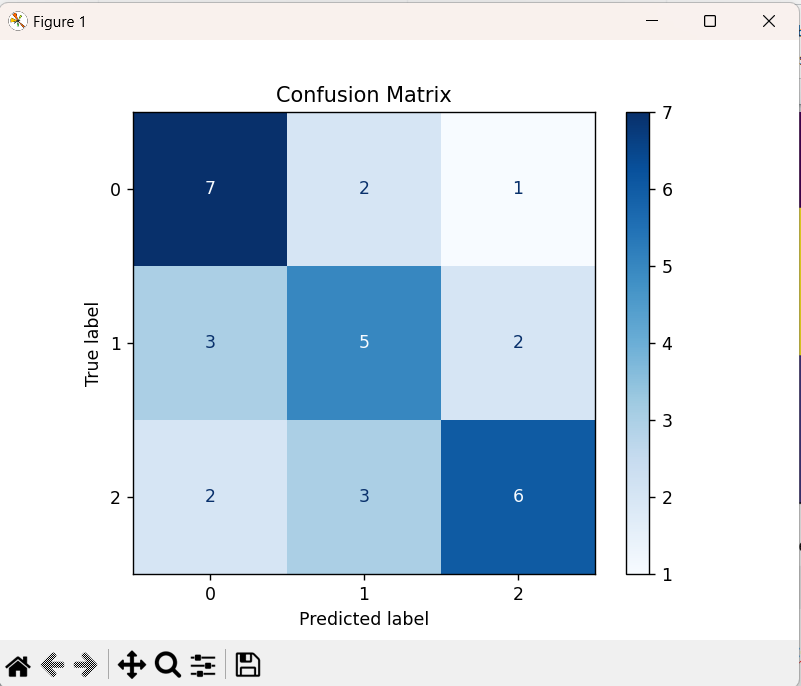


Slika 19) Grafik matrice konfuzije modela Logistic Regersion

* Perceptron



Slika 20) Kod I Accuracy od modela Perceptron



Slika 21) Grafik matrice konfuzije perceptron modela

# Literatura

* <https://talktechwithvipulgupta.medium.com/teaching-assistant-evaluation-classification-8088bba4ead0>
* <https://sci2s.ugr.es/keel/dataset.php?cod=188>
* <https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature>.
* <https://www.geeksforgeeks.org/decision-tree/>
* <https://www.datasciencecentral.com/logistic-regression-as-a-neural-network/>
* <https://www.javatpoint.com/perceptron-in-machine-learning>
* https://www.youtube.com/watch?v=qFJeN9V1ZsI&ab\_channel=freeCodeCamp.org