

Zadatak 1. VIŠESLOJNA POPTUNO POVEZANA NEURALNA MREŽA

```

from imblearn.over_sampling import SMOTE
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from keras.callbacks import EarlyStopping
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score, f1_score, \
    accuracy_score
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import keras.tuner as kt
from keras import models
from keras import layers

from keras import optimizers
from keras.regularizers import l2

```

Naš zadatak je da, na osnovu baze podataka **Genres.csv**, projektujemo **višeslojnu potpuno povezanu neuronsku mrežu (MLP)** koja vrši klasifikaciju muzičkih žanrova u tri klase: **Rap, Pop i RnB**.

Baza sadrži ukupno **2820 opservacija**, od kojih svaka poseduje **11 atributa** koji kvantitativno opisuju karakteristike pesme:

danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo.

Cilj neuronske mreže je da, na osnovu ovih atributa i njihovih međusobnih odnosa, nauči obrasce koji omogućavaju klasifikaciju pesama u pripadajuće muzičke žanrove:

- **0 – Rap** (1182 opservacije)
- **1 – RnB** (295 opservacija)
- **2 – Pop** (1343 opservacije)

Predobrada podataka

Prva faza u rešavanju problema je **predobrada podataka (preprocesiranje)**.

Za razliku od nekih drugih skupova, u bazi Genres.csv **ne postoje nedostajuće vrednosti (NaN)**, tako da nije bilo potrebe za njihovim uklanjanjem ili popunjavanjem.

U prvom koraku izvršeno je **učitavanje podataka Genres.csv** uz pomoć biblioteke *pandas*. U ovoj bazi nalaze se opservacije sa atributima koji opisuju muzičke žanrove, zajedno sa ciljnim atributom *genre*.

Kako bi neuronska mreža mogla da radi sa ciljnim atributom, bilo je neophodno izvršiti njegovo **mapiranje u numeričke vrednosti**. Konkretno, žanrovi su kodirani na sledeći način:

Rap → 0 *Pop* → 1 *RnB* → 2

Nakon toga, izdvojeni su **ulazni atributi**:

danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo.

Oni čine skup podataka ulaz, dok izlazni atribut (izlaz) predstavlja kodiranu vrednost kolone *genre*.

Ulazne promenljive predstavljaju niz numeričkih karakteristika koje mreža koristi za učenje, dok je izlazna promenljiva klasa žanra.

```
genre = pd.read_csv(r'C:\Users\PC\Desktop\nm\nm projekat\Genres.csv')
redosled = [0, 1, 2]
genre['genre'] = genre['genre'].map({'Rap': 0, 'Pop': 1, 'RnB': 2})
ulaz = genre[['danceability', 'energy', 'key', 'loudness', 'mode',
              'speechiness', 'acousticness', 'instrumentalness',
              'liveness', 'valence', 'tempo']]
izlaz = genre['genre']
print(izlaz.shape)
```

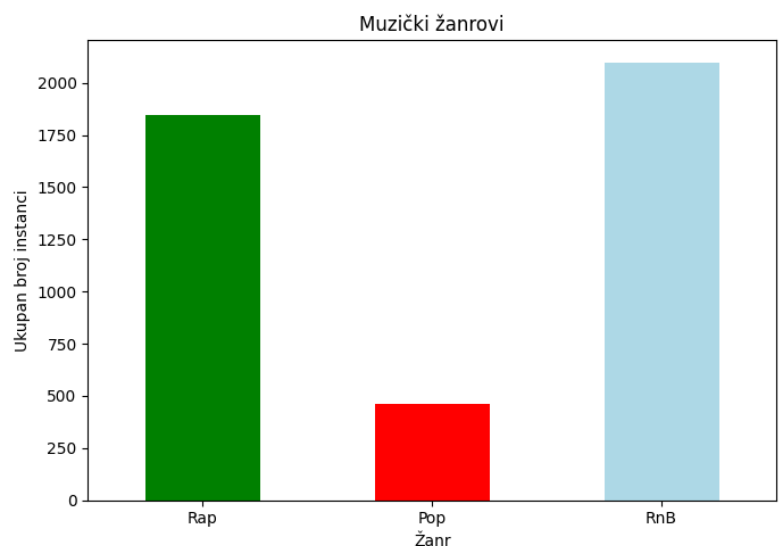
Histogram raspodele odbiraka po klasama

```
instance = genre['genre'].value_counts().reindex(redosled)
plt.figure(figsize=(7,5))
slika = instance.plot(kind='bar', color=['green', 'red', 'lightblue'])
plt.title('Muzički žanrovi')
plt.xlabel('Žanr')
plt.ylabel('Ukupan broj instanci')
slika.set_xticklabels(['Rap', 'Pop', 'RnB'], rotation=0)
plt.tight_layout()
plt.show()
```

Raspodela odbiraka po klasama

Rezultati pokazuju da je distribucija klasa **neuravnotežena**. Najviše instanci pripada žanru *Rap*, zatim slede *Pop* i *RnB* sa značajno manjim brojem primera.

Kao što se može primetiti na slici, klase **nisu** dovoljno **balansirane** za dalji rad sa njima. Moramo ih prvo izbalansirati.



Podela, balansiranje i normalizacija podataka

Ceo skup podataka podeljen je na **trening, validacioni i test skup**, jer performanse modela nije dovoljno procenjivati samo na trening podacima. Često se dešava da je estimator vrlo precizan na trening skupu, a daje loše rezultate na test skupu, zbog preprilagođavanja (*overfitting*).

Prva podela izdvojila je **20% podataka za test skup**, dok je preostalih 80% podeljeno na trening i validacioni skup u odnosu 80:20. Pritom je korišćena **stratifikacija po klasama**, kako bi se u svim skupovima očuvala proporcionalna raspodela žanrova (*Rap, Pop, RnB*).

Zbog neuravnoteženosti klasa u originalnom skupu, izvršeno je **balansiranje trening skupa** korišćenjem metode **SMOTE**, čime su retke klase proširene sintetisanim instancama, a odnosi između klasa u trening skupu postali uravnoteženi.

SMOTE:

Primeni se samo na trening skup, da bi model video balansirane klase.

Nikada se ne primenjuje na validaciju ili test, jer bi to davalo nerealno dobre rezultate.

```
ulaz_trening, ulaz_test, izlaz_trening, izlaz_test = train_test_split(
    ulaz, izlaz, test_size=0.2, random_state=45, stratify=izlaz)
ulaz_trening, ulaz_validacija, izlaz_trening, izlaz_validacija = train_test_split(
    ulaz_trening, izlaz_trening, train_size=0.8, stratify=izlaz_trening)

smote = SMOTE(random_state=45)
ulaz_trening_bal, izlaz_trening_bal = smote.fit_resample(ulaz_trening, izlaz_trening)
```

Dalje, radi jednostavnijeg i stabilnijeg obučavanja mreže, podaci su **normalizovani** pomoću StandardScaler, tako da su svi atributi skalirani u isti numerički opseg (standardizacija na sredinu 0 i standardnu devijaciju 1). Normalizacija je sprovedena **isključivo prema ekstremnim vrednostima trening skupa**, kako validacioni i test skup ne bi imali apriorna znanja o skaliranju. U kontekstu neuralnih mreža, normalizacija znači **pretvaranje svih ulaznih vrednosti u približno isti opseg**.

```
# NORMALIZACIJA

scaler = StandardScaler().fit(ulaz_trening_bal)
ulaz_trening_norm = scaler.transform(ulaz_trening_bal)
ulaz_validacija_norm = scaler.transform(ulaz_validacija)
ulaz_test_norm = scaler.transform(ulaz_test)

ulaz_trening_norm = ulaz_trening_norm.astype(np.float32)
izlaz_trening_bal = izlaz_trening_bal.astype(np.int32)
ulaz_validacija_norm = ulaz_validacija_norm.astype(np.float32)
izlaz_validacija = izlaz_validacija.astype(np.int32)
ulaz_test_norm = ulaz_test_norm.astype(np.float32)
izlaz_test = izlaz_test.astype(np.int32)
```

Na kraju, izvršena je provera **raspodele instanci po klasama u svim skupovima** (trening, validacija i test). Bar grafikon prikazuje broj uzoraka svake klase u svakom skupu, što omogućava vizuelnu potvrdu da su klase proporcionalno zastupljene nakon podela i SMOTE-a.

```
# RASPODELA ODBIRAKA PO SKUPOVIMA ZA TRENING TEST I VALIDACIJU

counts = [
    (np.count_nonzero(izlaz_trening == 0),
     np.count_nonzero(izlaz_trening == 1),
     np.count_nonzero(izlaz_trening == 2)),

    (np.count_nonzero(izlaz_validacija == 0),
     np.count_nonzero(izlaz_validacija == 1),
     np.count_nonzero(izlaz_validacija == 2)),

    (np.count_nonzero(izlaz_test == 0),
     np.count_nonzero(izlaz_test == 1),
     np.count_nonzero(izlaz_test == 2))
]

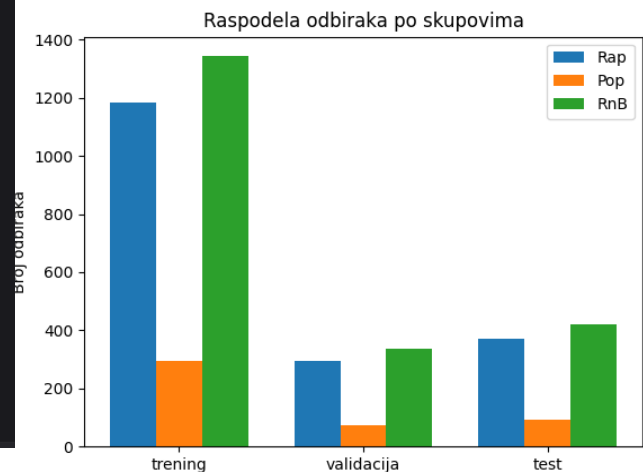
labels = ['trening', 'validacija', 'test']
rap = [c[0] for c in counts]
pop = [c[1] for c in counts]
rnb = [c[2] for c in counts]

x = np.arange(len(labels))
width = 0.25

fig, ax = plt.subplots()
ax.bar(x - width, rap, width, label='Rap')
ax.bar(x, pop, width, label='Pop')
ax.bar(x + width, rnb, width, label='RnB')

ax.set_ylabel('Broj odbiraka')
ax.set_title('Raspodela odbiraka po skupovima')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

plt.show()
```



Određivanje hiperparametara modela

Za optimizaciju performansi neuronske mreže korišćena je **random search metoda** za pretragu hiperparametara pomoću Keras Tuner biblioteke. Cilj je bio pronaći kombinaciju broja neurona, funkcije aktivacije, koeficijenta regularizacije, stope dropout-a i brzine učenja (learning rate) koja daje najbolje rezultate na validacionom skupu, uz racionalnu upotrebu resursa.

Fiksni parametri koji nisu podložni pretrazi uključuju:

- **Kriterijumska funkcija:** sparse_categorical_crossentropy, jer problem ima tri klase (0, 1, 2).
- **Funkcija aktivacije izlaznog sloja:** softmax, koja omogućava da izlaz mreže predstavlja verovatnoće za svaku klasu.
- **Optimizator:** Adam, zbog adaptivnog podešavanja konstante učenja na osnovu momentuma i kvadrata gradijenata, što često ubrzava i stabilizuje konvergenciju.
-

U procesu pretrage hiperparametara, razmatrani su sledeći parametri:

- **Broj neurona u prvom sloju:** između 32 i 256, sa korakom 16.
- **Funkcija aktivacije u prvom sloju:** sigmoid, relu ili tanh.
- **L2 regularizacija (reg):** između 0.001 i 0.5, sa korakom 0.005.
- **Dropout sloj (drop):** između 0 i 0.8, sa korakom 0.05.
- **Brzina učenja (learning_rate):** od 1e-5 do 1e-2, sa korakom 1e-4.

Prilikom obučavanja, korišćen je **EarlyStopping callback** koji prati validacionu tačnost (val_accuracy) i prekida obučavanje ako se performanse ne poboljšavaju 5 epoha zaredom, čime se sprečava overfitting i štedi vreme.

Random search je izvršen sa maksimalno 16 pokušaja (max_trials=16), gde je svaki model obučavan do 100 epoha sa batch veličinom od 64. Rezultati su pokazali da najbolja validaciona tačnost (val_accuracy) do sada iznosi približno **0.696**, što je znatno bolje od inicijalnih modela.

Traženje optimalnih hiperparametara treniranja

```
##### HIPERPARAMETRI #####
def make_model(hp):
    model = models.Sequential()
    model.add(layers.Input((ulaz_trening_norm.shape[1],)))
    no_units = hp.Int('units', min_value=32, max_value=256, step=16)
    act = hp.Choice('activation', values=['sigmoid', 'relu', 'tanh'])
    reg = hp.Float('reg', 0.001, 0.5, 0.005)
    drop = hp.Float('drop', 0, 0.8, 0.05)
    lr = hp.Float('learning_rate', 1e-5, 1e-2, 1e-4)
    model.add(layers.Dense(units=no_units, activation=act))
    model.add(layers.Dense(16, activation='relu', kernel_regularizer=l2(reg)))
    model.add(layers.Dropout(drop))
    model.add(layers.Dense(3, activation='softmax')) #zbog klasa 0, 1, 2
    opt = optimizers.Adam(learning_rate=lr)
    model.compile(optimizer=opt,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

stop_early = EarlyStopping(monitor='val_accuracy',
                           mode='max',
                           patience=5,
                           restore_best_weights=True,
                           verbose=2)

tuner = kt.RandomSearch(make_model,
                        objective='val_accuracy',
                        overwrite=True,
                        max_trials=16)

tuner.search(ulaz_trening_norm, izlaz_trening_bal,
             epochs=100,
             batch_size=64,
             validation_data=(ulaz_validacija_norm, izlaz_validacija),
             callbacks=[stop_early],
             verbose=3)
```

Optimalan broj neurona u prvom skrivenom sloju: 192

Optimalna funkcija aktivacije u prvom skrivenom sloju: relu

Optimalan koeficijent regularizacije u drugom skrivenom sloju: 0.051000000000000004

Optimalan dropout rate: 0.15000000000000002

Optimalna konstanta obučavanja: 0.0009100000000000001

Epoch 19: early stopping

Restoring model weights from the end of the best epoch: 14.

Najbolja tačnost na validacionom skupu: 0.6539

Trial 16 Complete [00h 00m 02s]
val_accuracy: 0.5219858288764954

Best val_accuracy So Far: 0.6964539289474487
Total elapsed time: 00h 00m 30s

```
best_hyperparam = tuner.get_best_hyperparameters(num_trials=1)[0]

print('Optimalan broj neurona u prvom skrivenom sloju: ', best_hyperparam['units'])
print('Optimalna funkcija aktivacije u prvom skrivenom sloju: ', best_hyperparam['activation'])
print('Optimalan koeficijent regularizacije u drugom skrivenom sloju: ', best_hyperparam['reg'])
print('Optimalan dropout rate: ', best_hyperparam['drop'])
print('Optimalna konstanta obučavanja: ', best_hyperparam['learning_rate'])

best_hyperparam = tuner.get_best_hyperparameters(num_trials=1)[0]
best_model = tuner.hypermodel.build(best_hyperparam)

history = best_model.fit(
    ulaz_trening_norm,
    izlaz_trening_bal,
    epochs=50,
    batch_size=64,
    validation_data=(ulaz_validacija_norm, izlaz_validacija),
    callbacks=[stop_early],
    verbose=0
)

loss, acc = best_model.evaluate(ulaz_validacija_norm, izlaz_validacija, verbose=0)
print(f'Najbolja tačnost na validacionom skupu: {acc:.4f}')

plt.figure()
plt.plot(history.history['loss'], label='Loss trening')
plt.plot(history.history['val_loss'], label='Loss validacija')
plt.xlabel('Epoch')
plt.ylabel('Vrednost loss funkcije')
plt.title('Loss funkcija: trening i validacioni skup')
plt.legend()
plt.grid(True)
plt.show()
```

Optimalne vrednosti su dobijene pomoću metode

tuner.get_best_hyperparameters(), a njihova primena omogućava postizanje najboljih rezultata klasifikacije uz racionalnu upotrebu resursa.

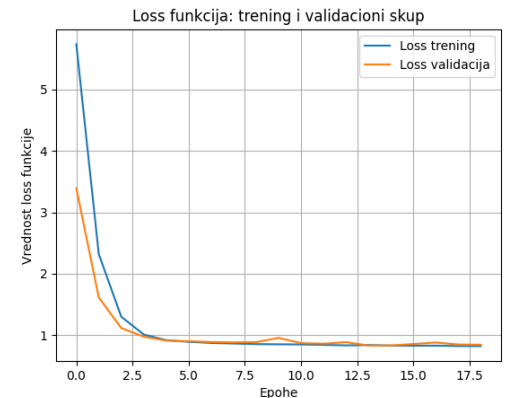
Prikaz performansi obučene neuralne mreže

Performanse modela

Model je obučavan 50 epoha uz batch veličinu 64.

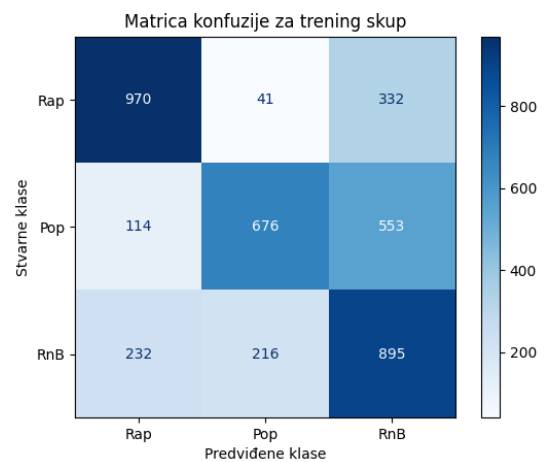
Na osnovu prikaza vrednosti *loss* funkcije kroz epohe za trening i validacioni skup, može se zaključiti da je obučavanje bilo stabilno, bez pojave preobučavanja.

Greška na trening skupu monotono opada, dok se na validacionom skupu stabilizuje, što ukazuje na uspešno učenje i adekvatno podešene hiperparametre.



```
y_pred = np.argmax(best_model.predict(ulaz_trening_norm, verbose=0), axis=1)
cm = confusion_matrix(izlaz_trening_bal, y_pred)

labels = ['Rap', 'Pop', 'RnB']
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap=plt.cm.Blues)
plt.title("Matrica konfuzije za trening skup")
plt.xlabel("Predviđene klase")
plt.ylabel("Stvarne klase")
plt.show()
```



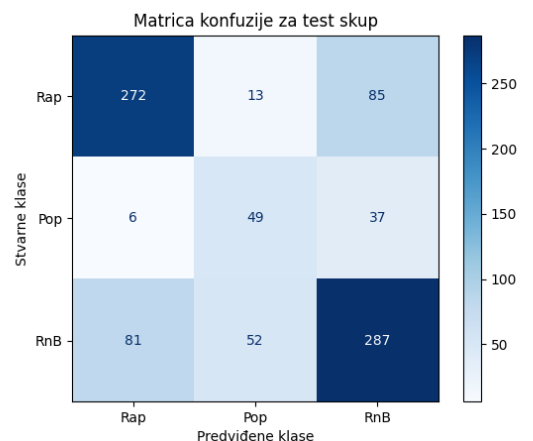
```
model = RandomForestClassifier(n_estimators=200, random_state=45)
model.fit(ulaz_trening_norm, izlaz_trening_bal)

y_pred = model.predict(ulaz_test_norm)
y_true = izlaz_test

acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred, average='macro')
sens = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')

print(f"Tačnost na test skupu: {acc*100:.2f} %")
print(f"Preciznost na test skupu: {prec*100:.2f} %")
print(f"Osetljivost na test skupu: {sens*100:.2f} %")
print(f"F1-score na test skupu: {f1*100:.2f} %")

cm = confusion_matrix(y_true, y_pred)
labels = ['Rap', 'Pop', 'RnB']
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap=plt.cm.Blues)
plt.title("Matrica konfuzije za test skup")
plt.xlabel("Predviđene klase")
plt.ylabel("Stvarne klase")
plt.show()
```



Nakon sprovedene obuke i evaluacije višeslojne potpuno povezane neuronske mreže za klasifikaciju muzičkih žanrova (*Pop*, *Rap* i *R&B*) iz skupa podataka **Genres.csv**, dobijeni su sledeći rezultati na test skupu:

Tačnost (Accuracy): 69.16 %
Preciznost (Precision): 62.25 %
Osetljivost (Recall): 64.66 %
F1-score: 63.09 %

Rezultati pokazuju da model ostvaruje **zadovoljavajuće performanse** u zadatku klasifikacije, iako postoji prostor za poboljšanje — naročito u pogledu preciznosti i osetljivosti, što sugerise da mreža povremeno pogrešno klasifikuje pojedine žanrove.

Razlike u broju opservacija po klasama (Pop – 1343, Rap – 1182, R&B – 295) verovatno su doprinele manjoj tačnosti u prepoznavanju manjinskih klasa.