Domaći zadatak iz predmeta Neuralne Mreže

Autori: Mladen Bašić (2018/011) i Damjan Denić (2018/0145)

In [ ]:

```
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.metrics import confusion_matrix, precision_recall_curve

torch.manual_seed(420)

import warnings
warnings.filterwarnings("ignore")
```

Postavka problema: Opstanak novih igrača u NBA ligi

Jedan od čestih problema menadžera timova u NBA ligi je procena da li je igrač koji je upravo došao u ligu ima potencijala da u njoj igra na duže staze. Menadžeri se pored mišljenja trenera i ostalih igrača vrlo često oslanjaju na analizu statistike igrača. U ovom radu predlažemo model koji će na osnovu statistike prve sezone igrača u NBA generisati predlog da li je vredno zadržati igrača u timu.

Treniranje i evaluacija vršena je na datasetu 5 Year Survival of NBA Rookies from 1980-2015 u kojem je računata isplativost zadržavanja igrača u timu ako on i dalje ima ugovor nakon 5 godina po dolasku u ligu. Dataset sadrži sve draftovane igrače u periodu od 1980. do 2015, njihovu statistiku u prvoj sezoni, kao i to da li su ostali u ligi nakon 5 godina.

Istraživanje podataka i predprocesiranje

U nastavku su podaci učitani u *dataframe* i urađeno je osnovno predprocesiranje kako bi se smanjila dimenzionalnost problema. Svaka od odluka pri filtriranju baze biće objašnjena neposredno pre ili posle odgovarajućeg bloka koda.

In [ ]:

```
# Load .csv file into pandas dataframe
df = pd.read_csv("data/rookie_df.csv", low_memory=False)
df
```

Out[ ]:

	Unnamed: 0	Year Drafted	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	...	FT%	OREB	DREB	REB	AST
0	0	2013	70	34.5	16.7	6.1	15.1	40.5	0.8	3.0	...	70.3	1.4	4.8	6.2	6.3
1	1	2013	70	32.3	12.8	4.9	12.8	38.0	1.6	4.8	...	90.3	0.5	2.4	3.0	5.7
2	2	2013	80	31.1	13.8	4.9	11.7	41.9	0.9	2.8	...	78.0	0.5	3.6	4.1	4.1
3	3	2013	82	26.7	8.8	3.1	8.3	37.6	1.2	3.6	...	80.4	0.6	2.2	2.9	1.0
4	4	2013	77	24.6	6.8	2.2	5.4	41.4	0.5	1.5	...	68.3	1.0	3.4	4.4	1.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1419	1419	1980	59	10.1	3.2	1.4	3.1	45.4	0.0	0.1	...	56.4	0.5	0.6	1.0	1.2
1420	1420	1980	47	9.6	2.0	0.7	2.0	35.8	0.0	0.1	...	78.1	0.2	0.7	0.9	1.5

	Unnamed: 0	Year Drafted	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	...	FT%	OREB	DREB	REB	AST
1421	1421	1980	60	8.9	2.8	1.0	3.0	33.0	0.0	0.1	...	73.9	0.5	0.7	1.2	1.3
1422	1422	1980	55	8.4	2.7	1.0	2.5	41.2	0.0	0.1	...	43.5	0.5	1.2	1.7	0.3
1423	1423	1980	44	5.3	2.6	1.1	3.1	34.8	0.0	0.1	...	45.9	0.6	0.7	1.3	0.3

1424 rows × 23 columns

In [ ]:

```
# Print all column names
df.columns
```

Out[ ]:

```
Index(['Unnamed: 0', 'Year Drafted', 'GP', 'MIN', 'PTS', 'FGM', 'FGA', 'FG%',
       '3P Made', '3PA', '3P%', 'FTM', 'FTA', 'FT%', 'OREB', 'DREB', 'REB',
       'AST', 'STL', 'BLK', 'TOV', 'EFF', 'target'],
      dtype='object')
```

Odabir relevantnih atributa

U datasetu postoji 22 atributa za svakiog igraca, medju kojima postoje neki koji su očigledno korelisani. Uzmimo za primer slobodna bacanja: za svakog igrača je vođena statistika koliko bacanja je igrač šutira (`FGA`), koliko ih je pogodio (`FGM`), kao i procenat uspešnih bacanja (`FG%`). Očigledno je da je procenat moguće nedvosmisleno izračunati iz prva dva podatka, tako da je on otklonjen. Odluka da se zadže dva broja umesto procenta doneta je kako bi model mogao da uračuna situacije u kojima je igrač imao mali broj pokušaja i samim time procenat ne prenosi adekvatno njegovu stvarnu efikasnost.

U nastavku je zadržano 16 atributa i otklonjeni su svi igrači sa nedefinisanim poljima.

Atribut	Opis
GP	Broj odigranih utakmica
MIN	Prosečan broj minuta u igri
PTS	Prosečan broj postignutih poena
FGM	Prosečan broj postignutih šuteva iz igre
FGA	Prosečan broj pokušanih šuteva iz igre
3P Made	Prosečan broj postignutih šuteva za 3 poena
3PA	Prosečan broj pokušanih šuteva za 3 poena
FTM	Prosečan broj pogođenih slobodnih bacanja
FTA	Prosečan broj pokušanih slobodnih bacanja
OREB	Prosečan broj skokova u napadu
DREB	Prosečan broj skokova u odbrani
AST	Prosečan broj asistencija
STL	Prosečan broj ukradenih lopti
BLK	Prosečan broj blokiranih šuteva
TOV	Prosečan broj izgubljenih lopti
EFF	Prosečan skor efikasnosti igrača

In [ ]:

```
# Choose valuable features
clean_df = df[['GP', 'MIN', 'PTS', 'FGM', 'FGA',
        '3P Made', '3PA', 'FTM', 'FTA', 'OREB', 'DREB',
        'AST', 'STL', 'BLK', 'TOV', 'EFF', 'target']].copy()
clean_df.dropna(inplace=True)
```

In [ ]:

```
# Describe features
clean_df.describe()
```

Out[ ]:

	GP	MIN	PTS	FGM	FGA	3P Made	3PA	F
count	1424.000000	1424.000000	1424.000000	1424.000000	1424.000000	1424.000000	1424.000000	1424.0000
mean	61.131320	17.483216	6.735323	2.602247	5.799930	0.238062	0.740520	1.2950
std	16.828774	8.265126	4.324453	1.666493	3.545943	0.385139	1.057448	0.9819
min	11.000000	3.100000	0.700000	0.300000	0.800000	0.000000	0.000000	0.0000
25%	48.000000	10.900000	3.600000	1.400000	3.200000	0.000000	0.000000	0.6000
50%	64.000000	15.950000	5.500000	2.100000	4.800000	0.000000	0.200000	1.0000
75%	77.000000	23.000000	8.800000	3.400000	7.500000	0.300000	1.100000	1.7000
max	82.000000	40.900000	28.200000	10.200000	19.800000	2.300000	6.500000	7.7000

In [ ]:

```
# Calculate percentage of True datapoints
value_cnts = clean_df["target"].value_counts()
print(f"Dataset sadrži {len(clean_df)} igrača od kojih je {value_cnts[1] / len(clean_d
```

```
Dataset sadrži 1424 igrača od kojih je 60.11% ostalo u ligi nakon 5 godina.
```

Podela na trening i test

Dataset je podeljen na trening i test skup, od čega trening set sadrži 70% podataka. Korištena je fukncija `train_test_split` koja nasumično promeni mesta podataka u tabeli pre nego što uradi podelu na dva seta.

In [ ]:

```
from sklearn.model_selection import train_test_split

# Do train/test split
train_df, valid_df = train_test_split(clean_df, train_size=.7, random_state=420)
```

Normalizacija dataseta

Pošto se svi atributi u datasetu ne nalaze u istim opsezima, izvršena je normalizacija podataka. Za ovu svrhu upotrebljena je `StandardScaler` klasa, ima mogućnost fitovanja parametara normalizacije na jednom skupu, i odvojenu primenu iste. Parametri normalizacije su fitovani na trening setu, a onda primenjeni i na treningu i na testu. Na ovaj način nije došlo do curenja informacija iz testa u trening.

In [ ]:

```
from sklearn.preprocessing import StandardScaler

input_columns = clean_df.columns[:-1]

# Fit scaler
scaler = StandardScaler()
scaler.fit(train_df[input_columns].values)

# Normalize train data
norm_train_np = scaler.transform(train_df[input_columns].values)
norm_train_df = pd.DataFrame(norm_train_np, columns=input_columns)
norm_train_df["target"] = train_df["target"].values

# Normalize validation data
norm_valid_np = scaler.transform(valid_df[input_columns])
norm_valid_df = pd.DataFrame(norm_valid_np, columns=input_columns)
norm_valid_df["target"] = valid_df["target"].values
```

Ovime je završeno predprocesiranje podataka, i oni su spremni za treniranje modela.

Definisanje modela

Radi predikcije dugovečnosti igrača na osnovu statistike u prvoj godini implementirana je poptpuno povezana neuronska mreža. Mrežu je moguće inicilaizovati sa proizvoljnim brojem skrivenih slojeva, a postoji tačno jedan izlazni neuron. Izlaz modela predstavlja njegovo poverenje da će igrač opstati u ligi, u rasponu [0, 1].

Svi delovi generisanja modela, treninga i evaluacije dekomponovani su u odgovarajuće funkcije i klase radi jednostavnijeg eksperimentisanja kasnije.

Klasa za učitavanje podataka

Tokom treninga je poželjno da pri tokom svake epohe treninga, podaci budu prosleđeni modelu u različitim redosledima. `pytorch` obezbeđuje ovu funkcionalnost, kao i jednostavno čitanje podataka po *batch*-evima iteracijom kroz klasu `Dataloader`. Ova klasa pri svojoj inicijalizaciji zahteva objekat koji nasleđuje klasu `Dataset`.

`Dataset` je klasa koja sadrži sve podatke treninga/testa i potrebno je u njoj implementirati metod `__getitem__` koji vraća ulazne podatke kao i očekivanu vrednost za model, na osnovu indeksa u bazi. U nastavku je implementirana klasa `NBA_Dataset`, koja obezbeđuje ovu funkcionalnost za odgovarajući dataset.

In [ ]:

```
from torch.utils.data import Dataset, DataLoader

""""""
class NBA_Dataset(Dataset):
    """Class for holding data NBA rookie statistics, and if they remained in the league

    Args:
        Dataset (Dataset): Pytorch base dataset class.
    """

    def __init__(self, df: pd.DataFrame) -> None:
        """Initialize torch Dataset based on pandas Dataframe.

        Args:
            df (pd.DataFrame): Pandas dataframe with player first year statistics.
        """
        super(NBA_Dataset, self).__init__()

        # Take all except the last column as inputs
        self.inputs = df.values[:, :-1].astype(float)

        # Cast Targets column to numpy
        self.targets = df["target"].values.astype(int)

        # Save dataframe as part of the class
        self.df = df

        # Define dataset length
        self.len = len(df)

    def __getitem__(self, index) -> dict:
        """Return dict with information of data point at `index`

        Args:
            index (int): Index of specific data point in dataset.
```

```
        Returns:
            dict: Dictionary containing two fields
                        Inputs: Tensor of player statistics.
                        Targets: Boolean value for if the player remained in the leagu
        """
        # Get input
        input_row = torch.Tensor(self.inputs[index, :])

        # Get output
        target = torch.Tensor([self.targets[index]])

        return {
            "Inputs": input_row,
            "Targets": target,
        }

    def __len__(self):
        """Get dataset length."""
        return self.len
```

Nakon inicijalizacije `Dataset` klase, ona je *wrap*-ovana u `Dataloader` klasu. U okviru ove klase su definisane veličina *batch*-a, broj radnih jedinica, kao i opcija da li redosled podataka treba randomizovati pred svaku epohu.

In [ ]:

```
train_dataset = NBA_Dataset(norm_train_df)

train_parameters = {
    "batch_size": 64,
    "shuffle": True,
    "num_workers": 1,
}

training_loader = DataLoader(train_dataset, **train_parameters)

# Validation dataset
valid_dataset = NBA_Dataset(norm_valid_df)

valid_parameters = {
    "batch_size": 64,
    "shuffle": False,
    "num_workers": 1,
}

validation_loader = DataLoader(valid_dataset, **valid_parameters)
```

Arhitektura modela

Model je implementiran u klasi `NBA_Survival_Predictor`. Mrežu je moguće inicijalizovati sa proizvoljnim brojem skrivenih slojeva, čije se veličine prosleđuju konstruktoru u vidu liste. Model je takođe moguće inicijalizovati sa proizvoljnom aktivacionom funkciojom, a ako ona nije prosleđena, aktivaciona fukcija će biti hiperbolički tangens.

U okviru `forward` funkcije implementiran je prolazak kroz model. Podaci se provlače kroz sve inicijalizovane objekte u listi slojeva. Objekti su naizmenično potpuno povezani slojevi i aktivacione funkcije, a umesto poslednje aktivacione funkcije primenjen sigmoid. Sigmoidna funkcija je upotrebljena kao standardna aktivaciona funkcija izlaznog sloja, prilikom klasifikacionih problema.

In [ ]:

```
import torch.nn as nn

class NBA_Survival_Predictor(nn.Module):
    """Class for estimating chance of NBA rookie surviving in the league."""
```

```
    def __init__(self, layer_sizes: list, activation_function: nn.Module = None) -> Non
        """Initialize fully-connected model based on input parameters."""
        super(NBA_Survival_Predictor, self).__init__()

        # Check if input and output layers are passed
        assert len(layer_sizes) >= 2

        # Check if activation function is not passed
        if activation_function is None:
            activation_function = nn.ReLU()

        # Define activation function
        self.activation = activation_function

        # Initialize list of layers
        layer_list = []

        # Initialize each layer and activation function
        for in_size, out_size in zip(layer_sizes[:-1], layer_sizes[1:]):
            layer_list.append(nn.Linear(in_size, out_size))
            layer_list.append(nn.Tanh())

        # Cast layer list to nn.Module
        self.fc_layers = nn.ModuleList(layer_list)

    def forward(self, X) -> torch.Tensor:
        """Define network behavior when called on data."""
        # Pass data trough fully-connected layers
        # Skip last activation function
        for layer in self.fc_layers[:-1]:
            X = layer(X)

        # Calculate output probability as sigmoid of output
        output_probability = torch.sigmoid(X)

        return output_probability
```

Definisanje treninga i evaluacije

U nastavku se nalazi implementacija funkcije za treniranje i evaluaciju modela. Funkciji od ulaznih parametara očekuje model, optimizator i rečnik trenaznih objekata. Rečnik u sebi sadrži 4 polja, *loss* funkciju, *dataloader* za trening i validaciju, kao i broj epoha za trening.

U okviru funkcije, model je treniran po *batch*-evima, a zatim evaluiran. Na kraju svake epohe, sačuvani su vrednost *loss*-a i tačnost modela. Tačnost je odabrana kao glavna metrika performanse modela zato što je *dataset* balansiran.

In [ ]:

```
CLASS_THR = .5
def train_network(model: NBA_Survival_Predictor, optimizer: torch.optim.Optimizer, tra
    """Function trains and evaluates defined model using given optimizer on passed tra

    Returned values of the function are trained model, and a dictionary with training 
    metrics.

    Args:
        model (NBA_Survival_Predictor): Model for training.
        optimizer (torch.optim.Optimizer): Model optimizer.
        training_objects (dict): Dictionary containing loss function, number of epochs
                                 training and validation loaders.

    Returns:
        tuple: Passed model after training, and dictionary containing training and val
    """

    # Unpack dictionaries
```

```python
    loss_function = training_objects["Loss Function"]
    training_loader = training_objects["Train Dataloader"]
    validation_loader = training_objects["Valid Dataloader"]
    num_of_epochs = training_objects["Epochs"]

    # Initialize lists for logging training and validation metrics
    training_loss = []
    training_acc = []
    validation_loss = []
    validation_acc = []

    # Train model for set number of epochs
    for _ in range(num_of_epochs):

        # Reset training metrics
        train_epoch_loss = 0.0
        train_epoch_acc = 0.0

        # Set model in train mode
        model.train()

        # Loop over training set in batches
        for batch in training_loader:

            # Unpack batch
            X = batch["Inputs"]
            Y = batch["Targets"]

            # Reset optimizer gradients
            optimizer.zero_grad()

            # Run a forward pass
            probabilities = model(X)

            # Calculate loss
            loss = loss_function(probabilities, Y)
            train_epoch_loss += loss.item()

            # Update model
            loss.backward()
            optimizer.step()

            # Calculate predictions
            predictions = probabilities >= CLASS_THR

            # Log true predictions
            train_epoch_acc += sum(predictions == Y)

        # Calculate epoch metrics
        train_epoch_loss /= len(training_loader)
        train_epoch_acc /= len(training_loader.dataset)

        # Log epoch metrics
        training_loss.append(train_epoch_loss)
        training_acc.append(train_epoch_acc)

        # Reset validation metrics
        valid_epoch_loss = 0.0
        valid_epoch_acc = 0.0

        # Set model in evaluation mode
        model.eval()

        # Loop over validation set in batches
        for batch in validation_loader:

            # Unpack batch
            X = batch["Inputs"]
            Y = batch["Targets"]
```

```
            # Run forward pass
            probabilities = model(X)

            # Calculate loss
            loss = loss_function(probabilities, Y)
            valid_epoch_loss += loss.item()

            # Calculate predictions
            predictions = probabilities >= CLASS_THR

            # Log true predictions
            valid_epoch_acc += sum(predictions == Y)

        # Calculate validation metrics
        valid_epoch_loss /= len(validation_loader)
        valid_epoch_acc /= len(validation_loader.dataset)

        # Log validation metrics
        validation_loss.append(valid_epoch_loss)
        validation_acc.append(valid_epoch_acc)

    # Get metrics and prediction probabilities of final model
    probabilities = []
    predictions = []
    targets = []

    # Evaluate final model
    for batch in validation_loader:

        # Load batch
        X = batch["Inputs"]
        Y = batch["Targets"]

        # Run forward pass
        batch_probabilities = model(X)
        batch_predictions = batch_probabilities >= CLASS_THR

        # Log probabilities, predictions and targets
        probabilities.extend(batch_probabilities.detach().numpy().flatten().tolist())
        predictions.extend(batch_predictions.detach().numpy().flatten().tolist())
        targets.extend(Y.detach().numpy().flatten().tolist())

    # Cast to numpy array
    probabilities = np.array(probabilities)
    predictions = np.array(predictions)
    targets = np.array(targets)

    # Pack all results in a dictionary
    training_parameters = {
        "train loss": training_loss,
        "train acc": training_acc,
        "validation loss": validation_loss,
        "validation acc": validation_acc,
        "Prediction probabilities": probabilities,
        "Predictions": predictions,
        "Targets": targets,
    }

    return model, training_parameters
```

Treniranje različitih modela

U ovom odeljku je definisano tri modela iste arhitekture, ali sa drugačijim hiperparametrima. Za *loss* funkciju izabrana je **binarna krosentropija**, pošto rezultati predstavljaju klasifikaciju igrača u dve klase (opstaće/neće opstati u ligi).

Od optimizacionih funkcija, eksperimentisano je sa **stohastičkim gradijentalnim spustom** i **Adamovim optimizatorem**. Adam je pokazao konstantno bolje rezultate, pa je on korišten za sve primere.

Svi modeli su trenirani na istom datasetu, istim redosledom batcheva i na **400 epoha**.

U nastavku su definisane tri arhitekture: jedna koja je nedovoljno obučena, jedna koja je preobučena, i jedna koja je adekvatna. Slede definicije modela, njihovi treninzi i performanse.

In [ ]:
```
from torch.nn import BCELoss
from torch.optim import Adam, SGD

# Initialize loss function as Binary Cross Entropy
loss_function = BCELoss()

# Define learning rate
lr = 3e-4

# Define number of epochs
num_of_epochs = 400

# Wrap training loaders into dictionary
training_objects = {
    "Loss Function": loss_function,
    "Train Dataloader": training_loader,
    "Valid Dataloader": validation_loader,
    "Epochs": num_of_epochs,
}
```

Definicija, trening i evaluacija

Treniranje modela sa premalom arhitekturom

In [ ]:
```
# Define model with no hidden layers
underfit_model = NBA_Survival_Predictor([train_dataset.inputs.shape[1], 1])

# Define optimizer for model with 5 times smaller learning rate
underfit_optimizer = Adam(params=underfit_model.parameters(), lr=lr/5)

# Train and evaluate model
underfit_model, underfit_results = train_network(underfit_model, underfit_optimizer, t
```

Treniranje modela sa prevelikom arhitekturom

In [ ]:
```
# Define model with 2 hidden layers
overfit_model = NBA_Survival_Predictor([train_dataset.inputs.shape[1], 20, 4, 1])

# Define optimizer for model with 3 times bigger learning rate
overfit_optimizer = Adam(params=overfit_model.parameters(), lr=lr*3)

# Train and evaluate model
overfit_model, overfit_results = train_network(overfit_model, overfit_optimizer, train
```

Treniranje modela sa adekvatnom arhitekturom

In [ ]:
```
# Define model with 1 hidden layer
overfit_model = NBA_Survival_Predictor([train_dataset.inputs.shape[1], 15, 1])

# Define optimizer for model with original learning rate
overfit_optimizer = Adam(params=overfit_model.parameters(), lr=lr)
```

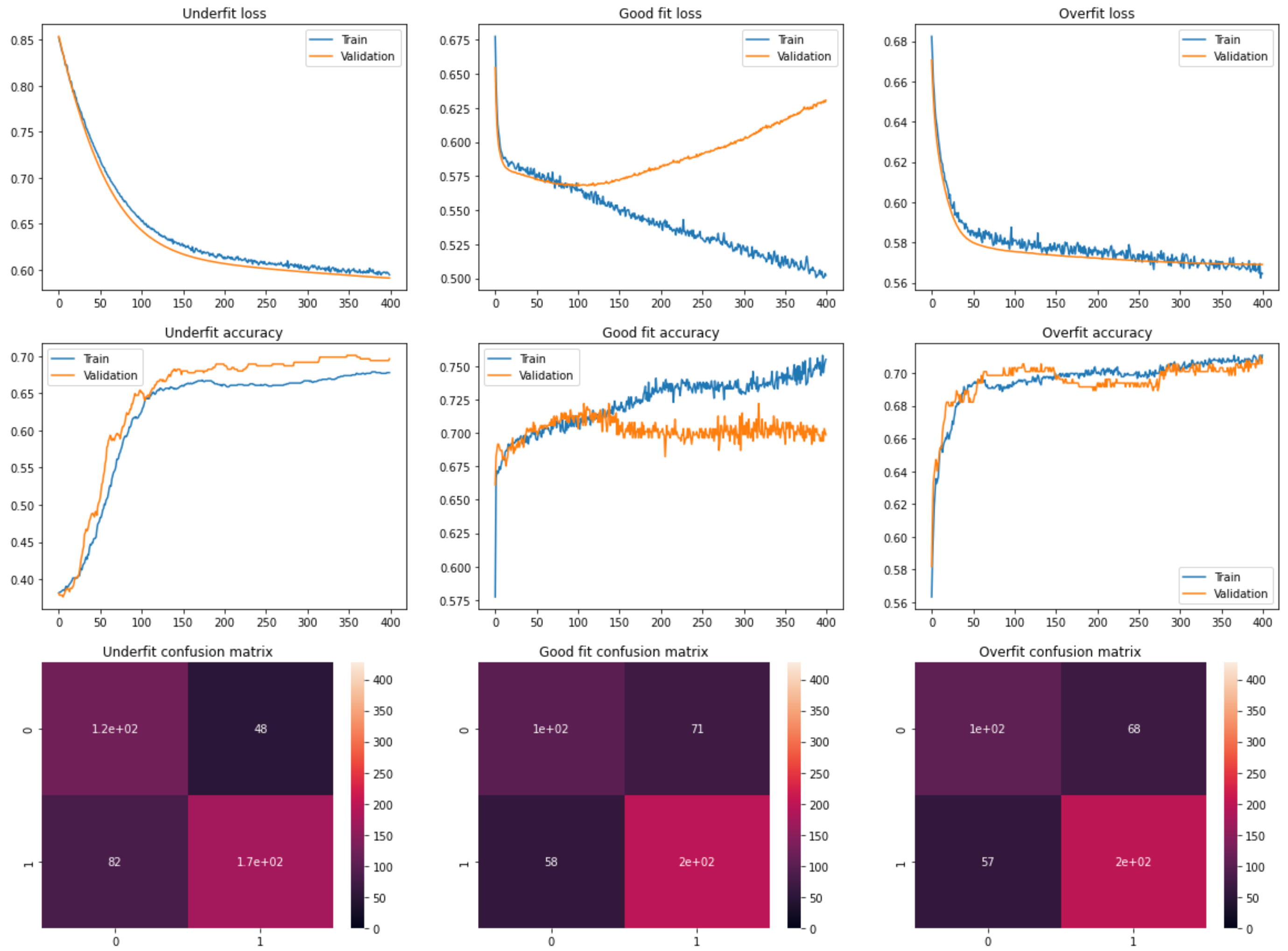
```
# Train and evaluate model
final_model, final_results = train_network(overfit_model, overfit_optimizer, training_(
```

Prikaz rezultata treninga i validacije

Prikazani su rezultati za sva tri modela. U prvom redu su prikazane vrednosti *loss* funkcije po epohama, za validaciju i trening. U drugom redu su prikazane tačnosti modela na treningu i validaciji po epohama. U trećem redu grafika su postavljene konfuzione matrice na validacionom setu na kraju treninga.

In [ ]:

```
# Initialize 3x3 subplots
fig, ax = plt.subplots(nrows=3, ncols=3, figsize=(20, 15))

# Initialize graph column names
title_labels = ["Underfit", "Good fit", "Overfit"]

# Loop over each column and generate graphs
for idx, results in enumerate([underfit_results, overfit_results, final_results]):

    # Plot loss
    ax[0, idx].plot(results["train loss"], label="Train")
    ax[0, idx].plot(results["validation loss"], label="Validation")
    ax[0, idx].legend()
    ax[0, idx].set_title(f"{title_labels[idx]} loss")

    # Plot accuracy
    ax[1, idx].plot(results["train acc"], label="Train")
    ax[1, idx].plot(results["validation acc"], label="Validation")
    ax[1, idx].legend()
    ax[1, idx].set_title(f"{title_labels[idx]} accuracy")

    # Plot confusion matrix
    sb.heatmap(confusion_matrix(results["Targets"], results["Predictions"]), vmin=0, vm
    ax[2, idx].set_title(f"{title_labels[idx]} confusion matrix")
```



Na graficima se može primetiti karakteristično ponašanje za sva tri modela.

Model bez skrivenih slojeva nije dovoljno kompleksan da bi efikasno vršio predviđanja. Ovo se oslikava u bajesu modela, koji se primećuje u konstantnoj razlici između tačnosti na treningu i validaciji.

Model sa 2 skrivena sloja preobučava. Očigledan simptom preobučavanja su funkcije *loss*-a i tačnosti na validaciji i treningu, koje posle 100. epohe počinju sve više da se udaljavaju jedna od druge.

Model sa 1 skrivenim slojem pokazuje najbolje performanse. Funkcije treninga i validacije imaju isti oblik i polako rastu kroz epohe. *Loss* funkcije imaju željenu eksponencijalnu raspodelu.

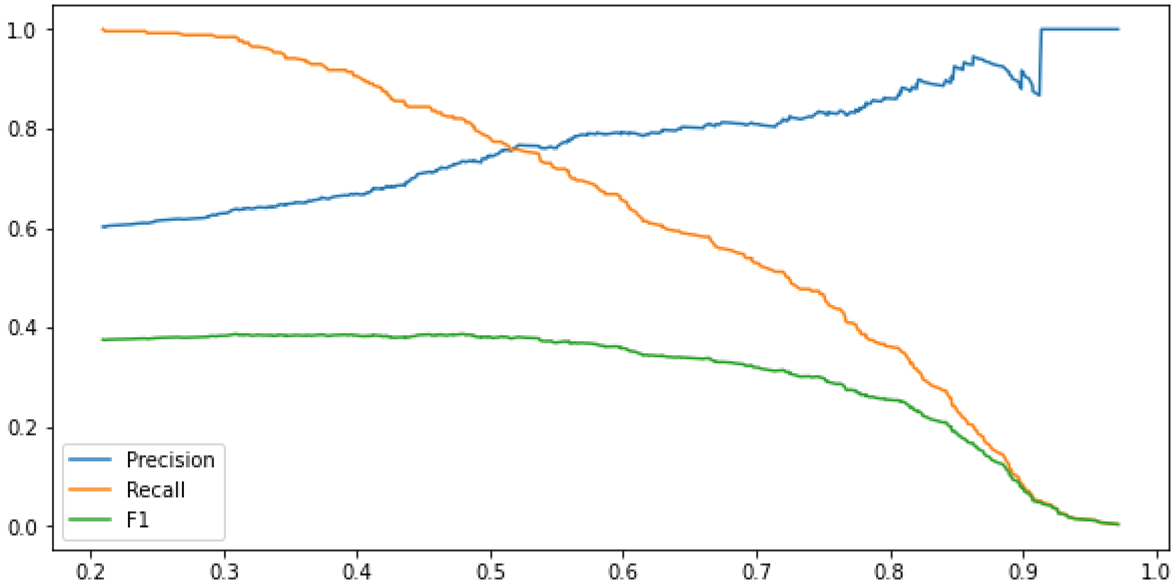
Zbog male veličine modela, trening za sva tri traje slično vreme, oko minut i 30 sekundi.

Preciznost/Odziv krive najboljeg modela

Nakon odabira najboljeg modela, menjan je prag odsecanja i analizirani su preciznost i odziv modela.

In [ ]:

```
# Generate precision and recall curve points
prec, recall, thrs = precision_recall_curve(final_results["Targets"], final_results["P

# Calculate f1-score
f1 = prec * recall / (prec + recall)

# Plot precision/recall/f1 curves
_, ax = plt.subplots(1, 1, figsize=(10, 5))
plt.plot(thrs, prec[:-1], label="Precision")
plt.plot(thrs, recall[:-1], label="Recall")
plt.plot(thrs, f1[:-1], label="F1")
plt.legend()
plt.show()
```



Funkcija preciznosti u rasponu od 0 do 0.5 raste sporije nego što funkcija odziva pada u istom segmentu. Kako je cilj modela da predvidi koje igrače treba zadržati u klubu, poželjan je visok odziv. Visok odziv odgovora pošto su igrači u prvim godinama karijere relativno jeftini za zadržavanje, a uvek ih je moguće menjati kasnije.

Sa druge strane, preciznost ne sme pasti značajno, kako klub ne bi plaćao igrače koji neće igrati na duže staze.

Kompromis između ova dva cilja možemo naći na pragu odsecanja od 0.45. Ovim smanjenjem praga će tačnost pasti, ali će odziv skočiti. Ova promena obezbeđuje da model zadržava više igrača koji će igrati u NBA.

In [ ]:

```
# Define final threshold
final_threshold = 0.45

# Get index of threshold with value of final threshold
thr_idx = sum(thrs <= final_threshold)

# Get best model precision and recall
final_precision = prec[thr_idx]
final_recall = recall[thr_idx]
final_f1 = f1[thr_idx]

# Calculate accuracy with final threshold
final_accuracy = sum(final_results["Targets"] == (final_results["Prediction probabilit
final_accuracy /= len(final_results["Targets"])

# Print final results
print("Statistika finalnog modela:")
print("---------------------------")
print(f"\tTačnost:    {final_accuracy * 100:.1f}%")
print(f"\tPreciznost: {final_precision * 100:.1f}%")
print(f"\tOdziv:      {final_recall * 100:.1f}%")
print(f"\tF1 skor:    {final_accuracy * 100:.1f}%")
```

```
Statistika finalnog modela:
---------------------------
        Tačnost:    70.1%
        Preciznost: 71.1%
        Odziv:      84.4%
        F1 skor:    70.1%
```

Automatsko traženje hyperparametara

Nakon ručnog odabira hiperparametara, implementirana je automatska pretraga. U ovu svrhu upotrebljena je biblioteka `optuna` , koju je koristiti sa već implementiranim modelom bez izmena.

Biblioteka radi generisanjem *study* sesije, kojoj treba zadati fukciju koju treba optimizovati. Ova funkcija se definiše odvojeno, i u njoj je moguće generisati hiperparametre iz proizvoljne raspodele i nakon toga trenirati tako definisan model.

Study sesiji se takođe prosleđuje i sampler. Cilj samplera je blago menjanje funkcija raspodela za odabir hiperparametara u zavisnosti od njihove performanse tokom sesije.

In [ ]:

```
import optuna
```

Funkcija cilja

U funkciji cilja su birani *learning rate*, broj skrivenih slojeva, kao i veličina svakog od skrivenih slojeva. *Learning rate* je biran iz raspodele koja je uniformna na logaritamskoj skali u rasponu od 10^{-6} do 10^{-4}. Broj skrivenih slojeva je biran nasumično između 1, 2 i 3, a svaki layer ima između 3 i 20 neurona.

Svaki model je treniran na 500 epoha.

Rezultat funkcije cilja je maksimalna tačnost na validaciji modela.

In [ ]:

```
def objective_function(trial: optuna.trial.Trial) -> float:
    """Objective function for NBA_Survival_Predictor training.

    Args:
        trial (optuna.trial.Trial): Optuna trial object.

    Returns:
        float: Maximal validation accuracy of the model.
    """
    # Pick learning rate from loguniform distribution
    lr = trial.suggest_loguniform("learning_rate", 1e-6, 1e-4)

    # Pick number of layers
    num_of_layers = trial.suggest_int("num_of_layers", 1, 3)

    # Initialize layer size list with input layer size
    layer_sizes = [train_dataset.inputs.shape[1]]

    # For each layer pick its size
    for idx in range(num_of_layers):
        layer_sizes.append(trial.suggest_int(f"layer_{idx}", 3, 20))

    # Add output layer size to layer sizes list
    layer_sizes.append(1)

    # Init model
    model = NBA_Survival_Predictor(layer_sizes)

    # Init optimizer
    optimizer = Adam(model.parameters(), lr=lr)

    # Set number of epochs
    training_objects["Epochs"] = 500

    # Train and evaluate model
    _, results = train_network(model, optimizer, training_objects)

    # Return best performance
    return max(results["validation acc"])
```

Traženje najboljih hiperparametara

U nastavku je kreiran *study* koji pokušava da maksimizuje prethodnu funkciju cilja. Definisan je maksimum od 100 pokušaja za maksimizaciju funkcije.

In [ ]:

```
study = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler()
study.optimize(objective_function, n_trials=100)
```

```
[I 2022-07-04 18:09:45,200] A new study created in memory with name: no-name-5438667a-21e3-4bcd-ae61-bb28e1382d50
[I 2022-07-04 18:11:28,971] Trial 0 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 6.828570120215091e-05, 'num_of_layers': 1, 'layer_0': 12}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:13:15,277] Trial 1 finished with value: 0.6915887594223022 and parameters: {'learning_rate': 4.1326746905280866e-05, 'num_of_layers': 3, 'layer_0': 14, 'layer_1': 7, 'layer_2': 11}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:14:59,863] Trial 2 finished with value: 0.6612149477005005 and parameters: {'learning_rate': 3.175638499390356e-06, 'num_of_layers': 2, 'layer_0': 11, 'layer_1': 12}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:16:42,397] Trial 3 finished with value: 0.6495327353477478 and parameters: {'learning_rate': 9.705883746747512e-06, 'num_of_layers': 1, 'layer_0': 7}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:18:30,958] Trial 4 finished with value: 0.6985981464385986 and parameters: {'learning_rate': 1.4222727853972257e-05, 'num_of_layers': 2, 'layer_0': 20, 'layer_1': 10}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:20:18,132] Trial 5 finished with value: 0.6915887594223022 and parameters: {'learning_rate': 8.910509522020181e-06, 'num_of_layers': 3, 'layer_0': 19, 'layer_1': 9, 'layer_2': 17}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:22:02,321] Trial 6 finished with value: 0.6822429895401001 and parameters: {'learning_rate': 6.2242405607625925e-06, 'num_of_layers': 3, 'layer_0': 10, 'layer_1': 3, 'layer_2': 17}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:23:47,156] Trial 7 finished with value: 0.5981308221817017 and parameters: {'learning_rate': 1.0097429295197447e-06, 'num_of_layers': 3, 'layer_0': 13, 'layer_1': 12, 'layer_2': 4}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:25:28,461] Trial 8 finished with value: 0.6658878326416016 and parameters: {'learning_rate': 1.5141496509889937e-05, 'num_of_layers': 1, 'layer_0': 6}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:27:14,477] Trial 9 finished with value: 0.6985981464385986 and parameters: {'learning_rate': 3.476360946242657e-06, 'num_of_layers': 3, 'layer_0': 19, 'layer_1': 12, 'layer_2': 10}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:28:57,267] Trial 10 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 9.359889177845498e-05, 'num_of_layers': 1, 'layer_0': 16}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:30:39,269] Trial 11 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 8.729735025232422e-05, 'num_of_layers': 1, 'layer_0': 15}. Best is trial 0 with value: 0.7009345889091492.
[I 2022-07-04 18:32:20,908] Trial 12 finished with value: 0.7056074738502502 and parameters: {'learning_rate': 7.518776360257204e-05, 'num_of_layers': 1, 'layer_0': 16}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:34:02,891] Trial 13 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 3.712270648484906e-05, 'num_of_layers': 2, 'layer_0': 3, 'layer_1': 20}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:35:44,614] Trial 14 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 3.668042798318718e-05, 'num_of_layers': 1, 'layer_0': 17}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:37:24,792] Trial 15 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 5.6794889832911435e-05, 'num_of_layers': 1, 'layer_0': 9}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:39:06,446] Trial 16 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 2.3566606730781953e-05, 'num_of_layers': 2, 'layer_0': 13, 'layer_1': 20}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:40:44,836] Trial 17 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 6.686902282786332e-05, 'num_of_layers': 1, 'layer_0': 17}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:42:23,345] Trial 18 finished with value: 0.6962617039680481 and parameters: {'learning_rate': 2.480765460053017e-05, 'num_of_layers': 2, 'layer_0': 12, 'laye
```

```
r_1': 16}. Best is trial 12 with value: 0.7056074738502502.
[I 2022-07-04 18:44:00,996] Trial 19 finished with value: 0.7079439163208008 and parame
ters: {'learning_rate': 2.2607398790168245e-05, 'num_of_layers': 1, 'layer_0': 9}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:45:39,403] Trial 20 finished with value: 0.6799065470695496 and parame
ters: {'learning_rate': 2.1719323957385277e-05, 'num_of_layers': 2, 'layer_0': 7, 'laye
r_1': 3}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:47:17,041] Trial 21 finished with value: 0.6985981464385986 and parame
ters: {'learning_rate': 5.2576679791976175e-05, 'num_of_layers': 1, 'layer_0': 9}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:49:01,214] Trial 22 finished with value: 0.6799065470695496 and parame
ters: {'learning_rate': 3.109995655587287e-05, 'num_of_layers': 1, 'layer_0': 4}. Best
 is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:50:43,148] Trial 23 finished with value: 0.6985981464385986 and parame
ters: {'learning_rate': 9.768854279595969e-05, 'num_of_layers': 1, 'layer_0': 11}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:52:22,061] Trial 24 finished with value: 0.7032710313796997 and parame
ters: {'learning_rate': 6.015760214663329e-05, 'num_of_layers': 1, 'layer_0': 9}. Best
 is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:54:00,015] Trial 25 finished with value: 0.6915887594223022 and parame
ters: {'learning_rate': 1.5717267281309165e-05, 'num_of_layers': 1, 'layer_0': 9}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:55:41,945] Trial 26 finished with value: 0.6939252614974976 and parame
ters: {'learning_rate': 4.896143324214389e-05, 'num_of_layers': 1, 'layer_0': 6}. Best
 is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:57:22,899] Trial 27 finished with value: 0.6985981464385986 and parame
ters: {'learning_rate': 2.7418606483727985e-05, 'num_of_layers': 2, 'layer_0': 7, 'laye
r_1': 16}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 18:59:06,247] Trial 28 finished with value: 0.6962617039680481 and parame
ters: {'learning_rate': 1.8224526044985865e-05, 'num_of_layers': 1, 'layer_0': 10}. Bes
t is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:00:47,770] Trial 29 finished with value: 0.6962617039680481 and parame
ters: {'learning_rate': 6.687614359397398e-05, 'num_of_layers': 1, 'layer_0': 5}. Best
 is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:02:25,433] Trial 30 finished with value: 0.5981308221817017 and parame
ters: {'learning_rate': 6.653980870865302e-06, 'num_of_layers': 1, 'layer_0': 8}. Best
 is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:04:09,356] Trial 31 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 4.338352729333181e-05, 'num_of_layers': 1, 'layer_0': 14}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:05:49,232] Trial 32 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 7.18235266640252e-05, 'num_of_layers': 1, 'layer_0': 12}. Best
 is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:07:29,827] Trial 33 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 4.244982682908396e-05, 'num_of_layers': 2, 'layer_0': 11, 'laye
r_1': 6}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:09:09,648] Trial 34 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 7.301215350980027e-05, 'num_of_layers': 1, 'layer_0': 14}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:10:49,053] Trial 35 finished with value: 0.6985981464385986 and parame
ters: {'learning_rate': 6.616474797269749e-05, 'num_of_layers': 1, 'layer_0': 15}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:12:28,213] Trial 36 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 8.377674326636875e-05, 'num_of_layers': 1, 'layer_0': 17}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:14:07,397] Trial 37 finished with value: 0.6985981464385986 and parame
ters: {'learning_rate': 3.253047682219118e-05, 'num_of_layers': 1, 'layer_0': 13}. Best
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:15:47,753] Trial 38 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 5.3246651844910146e-05, 'num_of_layers': 2, 'layer_0': 15, 'lay
er_1': 16}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:17:26,901] Trial 39 finished with value: 0.6962617039680481 and parame
ters: {'learning_rate': 1.0902402464896096e-05, 'num_of_layers': 1, 'layer_0': 10}. Bes
t is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:19:06,443] Trial 40 finished with value: 0.6308411359786987 and parame
ters: {'learning_rate': 1.0587669731453238e-06, 'num_of_layers': 1, 'layer_0': 18}. Bes
t is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:20:45,824] Trial 41 finished with value: 0.6985981464385986 and parame
ters: {'learning_rate': 7.950837789602477e-05, 'num_of_layers': 1, 'layer_0': 14}. Best
```

```
is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:22:24,463] Trial 42 finished with value: 0.6915887594223022 and parameters: {'learning_rate': 5.1265300441002404e-05, 'num_of_layers': 1, 'layer_0': 9}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:24:04,019] Trial 43 finished with value: 0.7056074738502502 and parameters: {'learning_rate': 5.9592135884822845e-05, 'num_of_layers': 1, 'layer_0': 8}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:25:42,595] Trial 44 finished with value: 0.7032710313796997 and parameters: {'learning_rate': 6.074914353138567e-05, 'num_of_layers': 1, 'layer_0': 7}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:27:22,223] Trial 45 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 3.9537104798347556e-05, 'num_of_layers': 1, 'layer_0': 6}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:29:01,971] Trial 46 finished with value: 0.7032710313796997 and parameters: {'learning_rate': 9.812683488782027e-05, 'num_of_layers': 1, 'layer_0': 8}. Best is trial 19 with value: 0.7079439163208008.
[I 2022-07-04 19:30:43,639] Trial 47 finished with value: 0.7102803587913513 and parameters: {'learning_rate': 9.879567051615823e-05, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 14, 'layer_2': 4}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:32:26,566] Trial 48 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 7.769895729038792e-05, 'num_of_layers': 3, 'layer_0': 16, 'layer_1': 15, 'layer_2': 3}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:34:09,526] Trial 49 finished with value: 0.7079439163208008 and parameters: {'learning_rate': 2.77266337257216e-06, 'num_of_layers': 3, 'layer_0': 13, 'layer_1': 14, 'layer_2': 7}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:35:52,670] Trial 50 finished with value: 0.5981308221817017 and parameters: {'learning_rate': 2.8784246269815337e-06, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 14, 'layer_2': 7}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:37:35,208] Trial 51 finished with value: 0.6471962332725525 and parameters: {'learning_rate': 4.4006332871199e-06, 'num_of_layers': 3, 'layer_0': 12, 'layer_1': 18, 'layer_2': 7}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:39:17,951] Trial 52 finished with value: 0.5981308221817017 and parameters: {'learning_rate': 1.6961722357337484e-06, 'num_of_layers': 3, 'layer_0': 13, 'layer_1': 14, 'layer_2': 7}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:41:00,301] Trial 53 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 1.1141552623441881e-05, 'num_of_layers': 3, 'layer_0': 16, 'layer_1': 18, 'layer_2': 5}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:42:43,300] Trial 54 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 7.196185875851513e-06, 'num_of_layers': 3, 'layer_0': 14, 'layer_1': 10, 'layer_2': 13}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:44:23,627] Trial 55 finished with value: 0.4065420627593994 and parameters: {'learning_rate': 2.3286183082515438e-06, 'num_of_layers': 2, 'layer_0': 10, 'layer_1': 14}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:46:06,465] Trial 56 finished with value: 0.6962617039680481 and parameters: {'learning_rate': 8.405290311865197e-06, 'num_of_layers': 3, 'layer_0': 8, 'layer_1': 18, 'layer_2': 9}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:47:47,739] Trial 57 finished with value: 0.677570104598999 and parameters: {'learning_rate': 4.983579435636766e-06, 'num_of_layers': 2, 'layer_0': 20, 'layer_1': 8}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:49:29,704] Trial 58 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 1.8888361329633344e-05, 'num_of_layers': 3, 'layer_0': 15, 'layer_1': 13, 'layer_2': 14}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:51:09,195] Trial 59 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 1.2792633713599942e-05, 'num_of_layers': 2, 'layer_0': 11, 'layer_1': 11}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:52:49,524] Trial 60 finished with value: 0.6985981464385986 and parameters: {'learning_rate': 3.0145079869658078e-05, 'num_of_layers': 3, 'layer_0': 12, 'layer_1': 6, 'layer_2': 5}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:54:27,683] Trial 61 finished with value: 0.6985981464385986 and parameters: {'learning_rate': 9.097803817026579e-05, 'num_of_layers': 1, 'layer_0': 8}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:56:05,982] Trial 62 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 4.701842980888707e-05, 'num_of_layers': 1, 'layer_0': 10}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:57:43,549] Trial 63 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 9.992310149322498e-05, 'num_of_layers': 1, 'layer_0': 8}. Best is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 19:59:22,125] Trial 64 finished with value: 0.7056074738502502 and parameters: {'learning_rate': 7.657086473417465e-05, 'num_of_layers': 1, 'layer_0': 13}. Best
```

```
is trial 47 with value: 0.7102803587913513.
[I 2022-07-04 20:01:00,308] Trial 65 finished with value: 0.7126168012619019 and parame
ters: {'learning_rate': 7.459531244738663e-05, 'num_of_layers': 1, 'layer_0': 13}. Best
is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:02:39,187] Trial 66 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 6.239071112835441e-05, 'num_of_layers': 1, 'layer_0': 14}. Best
is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:04:20,606] Trial 67 finished with value: 0.7079439163208008 and parame
ters: {'learning_rate': 7.575215349458807e-05, 'num_of_layers': 3, 'layer_0': 13, 'laye
r_1': 17, 'layer_2': 8}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:06:02,678] Trial 68 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 7.765064434507159e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 17, 'layer_2': 8}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:07:44,416] Trial 69 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 7.166382023254619e-05, 'num_of_layers': 3, 'layer_0': 13, 'laye
r_1': 19, 'layer_2': 5}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:09:25,806] Trial 70 finished with value: 0.6939252614974976 and parame
ters: {'learning_rate': 3.6418308265719736e-05, 'num_of_layers': 3, 'layer_0': 11, 'lay
er_1': 15, 'layer_2': 3}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:11:08,438] Trial 71 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 8.126813489422613e-05, 'num_of_layers': 3, 'layer_0': 13, 'laye
r_1': 17, 'layer_2': 20}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:12:50,266] Trial 72 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 5.6362058747957576e-05, 'num_of_layers': 3, 'layer_0': 13, 'lay
er_1': 17, 'layer_2': 20}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:14:32,633] Trial 73 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 8.819231305949488e-05, 'num_of_layers': 3, 'layer_0': 13, 'laye
r_1': 17, 'layer_2': 18}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:16:14,944] Trial 74 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 5.7397073864969316e-05, 'num_of_layers': 3, 'layer_0': 15, 'lay
er_1': 13, 'layer_2': 13}. Best is trial 65 with value: 0.7126168012619019.
[I 2022-07-04 20:17:56,661] Trial 75 finished with value: 0.7196261882781982 and parame
ters: {'learning_rate': 8.741124426324322e-05, 'num_of_layers': 3, 'layer_0': 10, 'laye
r_1': 15, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:19:39,980] Trial 76 finished with value: 0.7102803587913513 and parame
ters: {'learning_rate': 8.402580547086235e-05, 'num_of_layers': 3, 'layer_0': 19, 'laye
r_1': 15, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:21:22,168] Trial 77 finished with value: 0.7126168012619019 and parame
ters: {'learning_rate': 4.519900635400331e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 15, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:23:03,435] Trial 78 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 4.747649001324971e-05, 'num_of_layers': 3, 'layer_0': 10, 'laye
r_1': 15, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:24:45,292] Trial 79 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 6.930420949064457e-05, 'num_of_layers': 3, 'layer_0': 11, 'laye
r_1': 13, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:26:27,064] Trial 80 finished with value: 0.7079439163208008 and parame
ters: {'learning_rate': 8.63910647605304e-05, 'num_of_layers': 3, 'layer_0': 12, 'layer
_1': 15, 'layer_2': 8}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:28:09,220] Trial 81 finished with value: 0.7126168012619019 and parame
ters: {'learning_rate': 9.953572139182872e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 15, 'layer_2': 8}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:30:23,245] Trial 82 finished with value: 0.7102803587913513 and parame
ters: {'learning_rate': 9.009400510409199e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 15, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:32:05,906] Trial 83 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 8.793618969526247e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 15, 'layer_2': 5}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:33:47,542] Trial 84 finished with value: 0.7102803587913513 and parame
ters: {'learning_rate': 9.027439551328004e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 14, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:35:31,235] Trial 85 finished with value: 0.7009345889091492 and parame
ters: {'learning_rate': 6.530251675229412e-05, 'num_of_layers': 3, 'layer_0': 9, 'layer
_1': 13, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:37:13,037] Trial 86 finished with value: 0.7056074738502502 and parame
ters: {'learning_rate': 9.528321611215541e-05, 'num_of_layers': 3, 'layer_0': 12, 'laye
r_1': 16, 'layer_2': 9}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:38:55,222] Trial 87 finished with value: 0.7102803587913513 and parame
ters: {'learning_rate': 8.588168945307693e-05, 'num_of_layers': 3, 'layer_0': 11, 'laye
```

```
r_1': 14, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:40:38,416] Trial 88 finished with value: 0.7079439163208008 and parameters: {'learning_rate': 8.623217549712777e-05, 'num_of_layers': 3, 'layer_0': 19, 'layer_1': 14, 'layer_2': 4}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:42:20,077] Trial 89 finished with value: 0.7009345889091492 and parameters: {'learning_rate': 6.85137524684722e-05, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 12, 'layer_2': 4}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:44:02,191] Trial 90 finished with value: 0.7126168012619019 and parameters: {'learning_rate': 9.658965811805751e-05, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 16, 'layer_2': 6}. Best is trial 75 with value: 0.7196261882781982.
[I 2022-07-04 20:45:43,852] Trial 91 finished with value: 0.7242990732192993 and parameters: {'learning_rate': 9.980320768931802e-05, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 16, 'layer_2': 6}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:47:25,058] Trial 92 finished with value: 0.7102803587913513 and parameters: {'learning_rate': 8.192564338948574e-05, 'num_of_layers': 3, 'layer_0': 10, 'layer_1': 16, 'layer_2': 6}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:49:06,702] Trial 93 finished with value: 0.6939252614974976 and parameters: {'learning_rate': 9.748416633810863e-05, 'num_of_layers': 3, 'layer_0': 10, 'layer_1': 16, 'layer_2': 4}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:50:48,226] Trial 94 finished with value: 0.7032710313796997 and parameters: {'learning_rate': 9.964467224544998e-05, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 16, 'layer_2': 5}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:52:29,538] Trial 95 finished with value: 0.7032710313796997 and parameters: {'learning_rate': 7.096439481601283e-05, 'num_of_layers': 3, 'layer_0': 12, 'layer_1': 15, 'layer_2': 7}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:54:16,110] Trial 96 finished with value: 0.7032710313796997 and parameters: {'learning_rate': 6.144067063013234e-05, 'num_of_layers': 3, 'layer_0': 10, 'layer_1': 15, 'layer_2': 6}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:56:03,686] Trial 97 finished with value: 0.6985981464385986 and parameters: {'learning_rate': 5.357305693014646e-05, 'num_of_layers': 3, 'layer_0': 11, 'layer_1': 14, 'layer_2': 7}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:57:51,237] Trial 98 finished with value: 0.7056074738502502 and parameters: {'learning_rate': 8.967060761128442e-05, 'num_of_layers': 3, 'layer_0': 12, 'layer_1': 16, 'layer_2': 3}. Best is trial 91 with value: 0.7242990732192993.
[I 2022-07-04 20:59:38,588] Trial 99 finished with value: 0.7032710313796997 and parameters: {'learning_rate': 7.939086754565047e-05, 'num_of_layers': 3, 'layer_0': 10, 'layer_1': 14, 'layer_2': 5}. Best is trial 91 with value: 0.7242990732192993.
```

Promenom hiperparametara dobijen je model koji postiže tačnost na validacionom setu od 72.5%.

Redukcija dimenzionalnosti

Kao poslednji pokušaj za unapređenje modela, implementirana je redukcija dimenzionalnosti *dataset*-a. Implementirana je **LDA** redukcija i set atributa je redukovan na 4. Redukcija je implementirana na normalizovanim podacima.

Nad ovakvim podacima treniran je perceptron, zbog malog broja atributa. *Learning rate* je menjan u `Optuna` *study*-u.

LDA redukcija

Kod ispod generiše kovariacionu matricu atributa i računa njihove sopstvene vektore. Od 4 vektora, koji odgovaraju atributima sa najvećim sopstvenim vrednostim, se formira matrica kojom ce biti transformisan *dataset*.

In [ ]:

```
# Calculate covariance matrix of attributes, without targets
cov_mat = norm_train_df.corr().to_numpy()[:-1, :-1]

# Calculate eigenvalues
eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)

# Take vector subset
```

```
num_of_components = 4
eigv_subset = eigen_vectors[:, -num_of_components:]
```

Transformišemo trening i test podatke dobijenom matricom.

In [ ]:

```
# Extract numpy matrix of training attributes
X = norm_train_df.to_numpy()[:, :-1]

# Calculate reduced set of attributes
X_reduced = np.dot(eigv_subset.T, X.T).T

# Initialize datafreme with reduced attributes
reduced_train_df = pd.DataFrame(X_reduced, columns=[f"feature {idx}" for idx in range(n

# Add target values to the dataframe
reduced_train_df["target"] = norm_train_df["target"]

# Print results
print("Redukovani trening dataframe:")
reduced_train_df
```

```
Redukovani trening dataframe:
```

Out[ ]:

	feature 0	feature 1	feature 2	feature 3	target
0	-0.075304	-1.715120	1.484751	-4.481953	1
1	1.448046	-1.503470	0.761977	-0.046641	0
2	-0.588629	-1.104397	0.809860	2.415575	0
3	-0.184169	1.746011	3.458867	-1.794428	0
4	-0.684922	-0.405477	-0.795244	1.132226	1
...	...	...	...	...	...
991	0.647024	-0.295353	-0.883091	1.752937	0
992	-0.415180	-0.542635	1.334532	0.409983	1
993	-0.473289	-0.036981	-0.757612	3.428555	1
994	1.342950	-0.138566	-1.769756	-5.330138	1
995	0.799782	-0.948564	-1.884324	-8.928293	1

996 rows × 5 columns

In [ ]:

```
# Extract numpy matrix of validation attributes
X = norm_valid_df.to_numpy()[:, :-1]

# Calculate reduced set of attributes
X_reduced = np.dot(eigv_subset.T, X.T).T

# Initialize datafreme with reduced attributes
reduced_valid_df = pd.DataFrame(X_reduced, columns=[f"feature {idx}" for idx in range(n

# Add target values to the dataframe
reduced_valid_df["target"] = norm_valid_df["target"]

# Print results
print("Redukovani validacioni dataframe:")
reduced_valid_df
```

```
Redukovani validacioni dataframe:
```

Out[ ]:

	feature 0	feature 1	feature 2	feature 3	target
0	0.842048	-0.399118	0.513916	2.476210	0

	feature 0	feature 1	feature 2	feature 3	target
1	-0.231093	0.047274	0.691117	-2.771577	1
2	1.533942	-0.479238	-1.632356	-6.701583	1
3	-0.207686	-2.813286	0.733615	-1.535932	0
4	0.530262	-1.971947	0.371842	-0.031465	0
...	...	...	...	...	...
423	-0.805139	-1.261224	0.959032	1.406696	1
424	-1.046820	-0.841364	1.052354	0.191309	0
425	-0.802015	-0.683596	0.214574	0.864434	1
426	-0.786885	-1.461726	0.373974	1.454171	0
427	1.644015	-1.795306	0.729423	-6.792128	1

428 rows × 5 columns

Generisanje *dataloader*-a

In [ ]:
```
reduced_train_dataset = NBA_Dataset(reduced_train_df)

train_parameters = {
    "batch_size": 64,
    "shuffle": True,
    "num_workers": 1,
}

reduced_training_loader = DataLoader(reduced_train_dataset, **train_parameters)

# Validation dataset
reduced_valid_dataset = NBA_Dataset(reduced_valid_df)

valid_parameters = {
    "batch_size": 64,
    "shuffle": False,
    "num_workers": 1,
}

reduced_validation_loader = DataLoader(reduced_valid_dataset, **valid_parameters)
```

Definisanje trening objekata

In [ ]:
```
# Initialize training objects dictionary
training_objects = {
    "Loss Function": loss_function,
    "Train Dataloader": reduced_training_loader,
    "Valid Dataloader": reduced_validation_loader,
    "Epochs": num_of_epochs,
}
```

Treniranje modela

Pošto je trenirani model perceptron, jedini hiperparametar koji poseduje je *learning rate*. Implementirana je nova funkcija cilja koja pretražuje optimalan *learning rate*.

In [ ]:
```
def objective_function_reduced(trial: optuna.trial.Trial) -> float:
    """Objective function for NBA_Survival_Predictor training, with reduced feature spa

    Args:
```

```python
        trial (optuna.trial.Trial): Optuna trial object.

    Returns:
        float: Maximal validation accuracy of the model.
    """
    # Pick learning rate from loguniform distribution
    lr = trial.suggest_loguniform("learning_rate", 1e-6, 1e-4)

    layer_sizes = [4, 1]

    # Init model
    model = NBA_Survival_Predictor(layer_sizes)

    # Init optimizer
    optimizer = Adam(model.parameters(), lr=lr)

    # Set number of epochs
    training_objects["Epochs"] = 300

    # Train and evaluate model
    _, results = train_network(model, optimizer, training_objects)

    # Return best performance
    return max(results["validation acc"])
```

In [ ]:

```python
study = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler())
study.optimize(objective_function_reduced, n_trials=5)
```

```
[I 2022-07-04 20:59:39,421] A new study created in memory with name: no-name-0834db16-5fea-4bc8-b566-8770493a8134
[I 2022-07-04 21:00:40,594] Trial 0 finished with value: 0.6004672646522522 and parameters: {'learning_rate': 1.0340115315748442e-06}. Best is trial 0 with value: 0.6004672646522522.
[I 2022-07-04 21:01:40,990] Trial 1 finished with value: 0.6214953064918518 and parameters: {'learning_rate': 3.6076527228153054e-05}. Best is trial 1 with value: 0.6214953064918518.
[I 2022-07-04 21:02:40,673] Trial 2 finished with value: 0.577102780342102 and parameters: {'learning_rate': 9.582401768738296e-06}. Best is trial 1 with value: 0.6214953064918518.
[I 2022-07-04 21:03:40,632] Trial 3 finished with value: 0.38785046339035034 and parameters: {'learning_rate': 2.54851827532148e-06}. Best is trial 1 with value: 0.6214953064918518.
[I 2022-07-04 21:04:40,706] Trial 4 finished with value: 0.39485982060432434 and parameters: {'learning_rate': 4.961718191455301e-06}. Best is trial 1 with value: 0.6214953064918518.
```

Model nakon redukcije dimenzionalnosti ima lošije performanse od početnog. Verovatan razlog ovoga je premali broj atributa, pa klase nisu separabilne.