Grafovske Neuronske Mreže - Klasifikacija čvorova

Jelisaveta Gavrilović i Marko Paunović

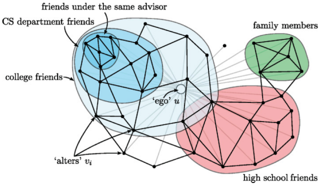
Matematički fakultet
Univerzitet u Beogradu

Beograd, 2024.

Pregled

Uvod

- Ego mreže - specifične strukture unutar društvenih mreža koje se fokusiraju na pojedinca (ego) i sve veze koje taj pojedinac ima s drugim korisnicima (čvorovima) unutar mreže
- Često se koriste u sociološkim, psihološkim i ekonomskim istraživanjima kako bi se analizirale socijalne interakcije i uticaji, kao i u razvoju preporučivačkih sistema, predikciji ponašanja korisnika i analizi zajednica u online platformama



Slika: Primer ego mreže

Uvod

- Skup podataka je preuzet sa sajta Stanford Network Analysis Project (SNAP), specifično iz Facebook podataka.
  - link: SNAP - Facebook
- Cilj projekta: klasifikacija korisnika u odnosu na pripadnost određenoj ego mreži
  - na osnovu analize interakcija i atributa korisnika, cilj je predvideti kojoj konkretnoj ego mreži korisnik pripada, uz mogućnost da jedan korisnik može biti deo jedne, nijedne ili više ego mreža

Podaci i struktura grafa

- Skup podataka: predstavljen u obliku grafa, gde su:
  - čvorovi korisnici
  - grane veze ili prijateljstva između dva korisnika
- neusmeren graf
  - ako je korisnik A prijatelj korisnika B, onda je i korisnik B prijatelj korisnika A

```
print(f"Broj čvorova u grafu: {G.number_of_nodes()}")
print(f"Broj ivica u grafu: {G.number_of_edges()}")
```

```
Broj čvorova u grafu: 4039
Broj ivica u grafu: 88234
```

Slika: Osnovne informacije o grafu

Podaci i struktura grafa

- Svaki korisnik ima i niz atributa koji opisuju njegove osobine i karakteristike
- Međutim, pošto je skup podataka skinut sa Facebook-a, svi atributi su anonimizovani Facebook-ovom internom reprezentacijom
    - možemo samo da uporedimo da li korisnici imaju neke iste karakteristike, ali ne i šta konkretno oni predstavljaju
- Svi atributi su binarni

```
print(f"Broj atributa: {len(G.nodes[0])}")
```

```
Broj atributa: 2255
```

Slika: Ukupan broj atributa za svaki čvor

Podaci i struktura grafa

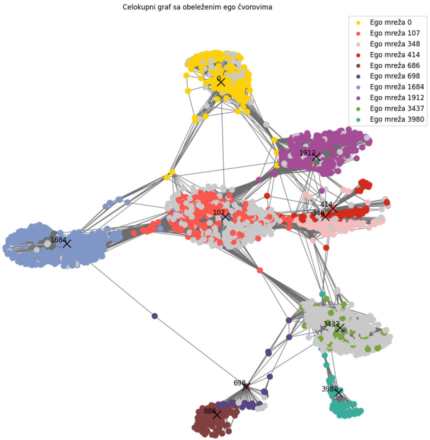
```
G.nodes[17]
```

```
{'0 birthday;anonymized feature 0': 1,
 '0 birthday;anonymized feature 1003': 0,
 '0 birthday;anonymized feature 1172': 0,
 '0 birthday;anonymized feature 2': 0,
 '0 birthday;anonymized feature 206': 0,
 '0 birthday;anonymized feature 208': 0,
 '0 birthday;anonymized feature 209': 0,
 '0 birthday;anonymized feature 376': 0,
 '0 birthday;anonymized feature 6': 0,
 '0 birthday;anonymized feature 729': 0,
 '1 birthday;anonymized feature 0': 0,
 '1 birthday;anonymized feature 1': 0,
 '1 birthday;anonymized feature 1004': 0,
 '1 birthday;anonymized feature 2': 0,
 '1 birthday;anonymized feature 207': 0,
 '1 birthday;anonymized feature 208': 0,
 '1 birthday;anonymized feature 3': 0,
 '1 birthday;anonymized feature 730': 0,
 '1 education;concentration;id;anonymized feature 14': 0,
 '10 birthday;anonymized feature 1006': 0,
 '10 birthday;anonymized feature 211': 0,
 '10 birthday;anonymized feature 7': 0,
 '10 education;classes;id;anonymized feature 10': 0,
 '10 education;concentration;id;anonymized feature 215': 0,
 '10 education;concentration;id;anonymized feature 339': 0,
 '10 education;concentration;id;anonymized feature 384': 0,
 '10 education;school;id;anonymized feature 370': 0,
 '10 education;year;id;anonymized feature 253': 0,
 '100 education;school;id;anonymized feature 446': 0,

        ...}
```

Slika: Atributi korisnika 17

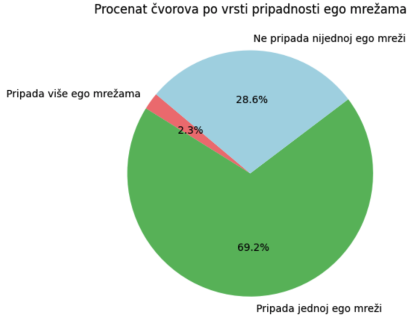
Vizuelizacija i analiza grafa



Slika: Struktura mreže sa obeleženim ego korisnicima i bojama za različite ego mreže

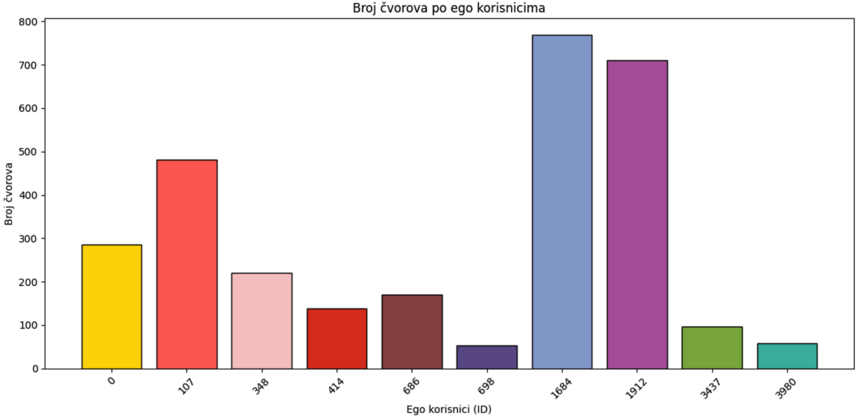
Vizuelizacija i analiza grafa

- Sivo obeleženi čvorovi na grafu, označavaju da čvor ne pripada nijednoj ego mreži
- Ukoliko čvorovi pripadaju više ego mrežama, onda su obeleženi bojom prve ego mreže



Slika: Procenat pripadnosti ego mrežama

Vizuelizacija i analiza grafa



Slika: Broj korisnika u svakoj ego mreži

GNN

- Graph Neural Networks - neuronske mreže koje su dizajnirane za rad s podacima koji su strukturirani u obliku grafova
  - koriste topološke informacije grafova i mogu modelirati složene odnose među čvorovima
- koriste se u raznim oblastima, uključujući biološke mreže, društvene mreže, finansije...

GCN

- Graph Convolutional Networks - specifičan tip GNN-a koji koristi konvolucione slojeve prilagođene za grafove
  - svaki čvor prikuplja informacije od svojih suseda, a onda čvor ažurira svoju reprezentaciju na osnovu prikupljenih podataka
  - ovaj proces se može ponavljati više puta, omogućavajući čvorovima da uče iz sve dubljih nivoa njihove okoline

Kod i analiza - GCN

- Kod se sastoji od nekoliko ključnih delova:
  - Učitavanje i priprema podataka
  - Definicija modela
  - Proces treniranja i evaluacije

Kod i analiza - GCN

- Učitavanje podataka

```
# ucitavanje grafa
with open('graph.pkl', 'rb') as f:
    G = pickle.load(f)
```

```
from torch_geometric.utils import from_networkx
data = from_networkx(G)
```

- Podela čvorova na trening, test i validacioni skup i kreiranje maski

```
# podela cvorova na trening, validacioni i test skup
nodes = list(G.nodes)
train_nodes, test_nodes = train_test_split(nodes, test_size=0.2, random_state=42)
train_nodes, val_nodes = train_test_split(train_nodes, test_size=0.2, random_state=42)
```

```
# kreiranje maske za skupove
train_mask = torch.tensor([True if node in train_nodes else False for node in range(len(G.nodes))])
val_mask = torch.tensor([True if node in val_nodes else False for node in range(len(G.nodes))])
test_mask = torch.tensor([True if node in test_nodes else False for node in range(len(G.nodes))])
```

Kod i analiza - GCN

- Podešavanje atributa i ciljnh promenljivih

```
# atributi
node_features = [list(G.nodes[node].values())[:-1] for node in G.nodes]
node_features = torch.tensor(node_features, dtype=torch.float)

# ciljne promenljive
labels = np.array([G.nodes[node]['target'] for node in G.nodes])
labels = torch.tensor(labels, dtype=torch.float)

# dodajemo i labelu koja ce da oznacava da cvor ne pripada nijednoj ego mrezi
# -> kako bismo mogli da dodelimo odgovarajucu tezinu zbog nebalansiranosti klasa :)
no_class_nodes = (labels.sum(dim=1) == 0).float()
dummy_class = no_class_nodes.unsqueeze(1)
labels = torch.cat([labels, dummy_class], dim=1)


data.x = node_features
data.y = labels
data.train_mask = train_mask
data.val_mask = val_mask
data.test_mask = test_mask
```

Kod i analiza - GCN

- Model

```
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels, dropout_rate):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.dropout = torch.nn.Dropout(dropout_rate)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_index)
        return x
```

Kod i analiza - GCN

- Funkcije za treniranje i testiranje modela

```
def train():
    model.train()
    optimizer.zero_grad()

    pred_train = model(data)[data.train_mask].to(device)
    loss = criterion(pred_train, data.y[data.train_mask].float().to(device))
    loss.backward()
    optimizer.step()

    preds_train = (torch.sigmoid(pred_train) > 0.5).float()
    acc_train = (preds_train == data.y[data.train_mask].to(device)).float().mean()
    precision_train = precision(pred_train.to(device), data.y[data.train_mask].to(device))
    recall_train = recall(pred_train.to(device), data.y[data.train_mask].to(device))
    f1_train = f1(pred_train.to(device), data.y[data.train_mask].to(device))

    return loss, acc_train.item(), precision_train.item(), recall_train.item(), f1_train.item()
```

```
def evaluate(mask):
    model.eval()
    with torch.no_grad():
        pred_val = model(data)[mask].to(device)
        loss_val = criterion(pred_val, data.y[mask].float().to(device))

        preds_val = (torch.sigmoid(pred_val) > 0.5).float()
        acc_val = (preds_val == data.y[mask].to(device)).float().mean()
        precision_val = precision(pred_val.to(device), data.y[mask].to(device))
        recall_val = recall(pred_val.to(device), data.y[mask].to(device))
        f1_val = f1(pred_val.to(device), data.y[mask].to(device))

        if torch.equal(mask, data.test_mask):
            # Vizualizacija matrice konfuzije na test skupu
            plot_confusion_matrix(data.y[data.test_mask].to(device), preds_val.to(device), labels.shape[1])


    return loss_val, acc_val.item(), precision_val.item(), recall_val.item(), f1_val.item()
```

Kod i analiza - GCN

- Metrike koje pratimo

```
# metrike
device = 'cuda' if torch.cuda.is_available() else 'cpu'
precision = torchmetrics.Precision(num_labels=labels.shape[1], average='macro', task='multilabel').to(device)
recall = torchmetrics.Recall(num_labels=labels.shape[1], average='macro', task='multilabel').to(device)
f1 = torchmetrics.F1Score(num_labels=labels.shape[1], average='macro', task='multilabel').to(device)
```

- Podešavanje hiperparametara

```
# Hiperparametri
hyperparameters = {
    'hidden_channels': [32, 64],
    'dropout_rate': [0.2, 0.5],
    'num_epochs': [50, 100]
}
learning_rate = 0.01
weight_decay = 5e-4
```

- Podešavanje težina klasa

```
train_labels = labels[data.train_mask]
nodes_in_ego_train = train_labels.sum(dim=0)

total_train_nodes_num = train_labels.shape[0]
class_weights = total_train_nodes_num / (nodes_in_ego_train + 1e-6)

print(f'Nodes in ego (train only): {nodes_in_ego_train}')
print(f'Class weights: {class_weights}')
```

```
Nodes in ego (train only): tensor([168., 290., 133.,  94., 112.,  33., 498., 474.,  68.,  36., 736.])
Class weights: tensor([15.3810,  8.9103, 19.4286, 27.4894, 23.0714, 78.3030,  5.1888,  5.4515,
        38.0000, 71.7778,  3.5109])
```

Kod i analiza - GCN

- Nakon treniranja modela za različite kombinacije hiperparametara i evaluacije na validacionom skupom, dobijeni su sledeci rezultati

```
Testing parameters: (32, 0.2, 50)
[Train] Epoch: 0, Loss: 1.3272, Accuracy: 0.6148, Precision: 0.0360, Recall: 0.3636, F1: 0.0635
[Train] Epoch: 10, Loss: 0.7498, Accuracy: 0.8488, Precision: 0.3346, Recall: 0.9891, F1: 0.4705
[Train] Epoch: 20, Loss: 0.3728, Accuracy: 0.9416, Precision: 0.6004, Recall: 0.9735, F1: 0.7120
[Train] Epoch: 30, Loss: 0.2760, Accuracy: 0.9433, Precision: 0.6274, Recall: 0.9763, F1: 0.7286
[Train] Epoch: 40, Loss: 0.2504, Accuracy: 0.9449, Precision: 0.6329, Recall: 0.9798, F1: 0.7339
[Validate] Loss: 0.2532, Accuracy: 0.9489, Precision: 0.6691, Recall: 0.9682, F1: 0.7523
Testing parameters: (32, 0.2, 100)
[Train] Epoch: 0, Loss: 1.3256, Accuracy: 0.5292, Precision: 0.0418, Recall: 0.4332, F1: 0.0720
[Train] Epoch: 10, Loss: 0.7102, Accuracy: 0.8623, Precision: 0.4264, Recall: 0.9850, F1: 0.5404
[Train] Epoch: 20, Loss: 0.3671, Accuracy: 0.9414, Precision: 0.6179, Recall: 0.9764, F1: 0.7216
[Train] Epoch: 30, Loss: 0.2770, Accuracy: 0.9446, Precision: 0.6386, Recall: 0.9730, F1: 0.7334
[Train] Epoch: 40, Loss: 0.2498, Accuracy: 0.9462, Precision: 0.6460, Recall: 0.9745, F1: 0.7414
[Train] Epoch: 50, Loss: 0.2329, Accuracy: 0.9475, Precision: 0.6595, Recall: 0.9780, F1: 0.7502
[Train] Epoch: 60, Loss: 0.2288, Accuracy: 0.9487, Precision: 0.6630, Recall: 0.9761, F1: 0.7536
[Train] Epoch: 70, Loss: 0.2209, Accuracy: 0.9513, Precision: 0.6724, Recall: 0.9807, F1: 0.7626
[Train] Epoch: 80, Loss: 0.2131, Accuracy: 0.9517, Precision: 0.6755, Recall: 0.9808, F1: 0.7661
[Train] Epoch: 90, Loss: 0.2106, Accuracy: 0.9519, Precision: 0.6806, Recall: 0.9798, F1: 0.7691
[Validate] Loss: 0.2503, Accuracy: 0.9518, Precision: 0.6860, Recall: 0.9686, F1: 0.7658
Testing parameters: (32, 0.5, 50)
[Train] Epoch: 0, Loss: 1.3256, Accuracy: 0.6140, Precision: 0.1450, Recall: 0.3682, F1: 0.0698
[Train] Epoch: 10, Loss: 0.7912, Accuracy: 0.7867, Precision: 0.2732, Recall: 0.9726, F1: 0.3878
[Train] Epoch: 20, Loss: 0.4492, Accuracy: 0.9106, Precision: 0.4498, Recall: 0.9676, F1: 0.5864
[Train] Epoch: 30, Loss: 0.3559, Accuracy: 0.9242, Precision: 0.5040, Recall: 0.9604, F1: 0.6316
[Train] Epoch: 40, Loss: 0.3023, Accuracy: 0.9320, Precision: 0.5412, Recall: 0.9751, F1: 0.6671
[Validate] Loss: 0.2705, Accuracy: 0.9486, Precision: 0.6689, Recall: 0.9773, F1: 0.7541
```

- Najbolji model je model sa hiperparametrima

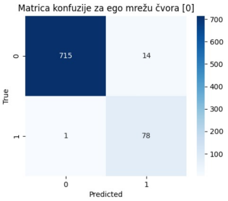
```
Best parameters: (64, 0.2, 100), with loss: 0.2450
```

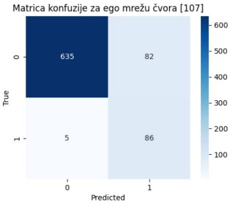
Kod i analiza - GCN

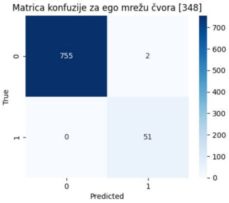
- Metrike na test skupu

```
Test Loss: 0.1904
Test Accuracy: 0.9557
Test Precision: 0.7121
Test Recall: 0.9727
Test F1 Score: 0.7896
```

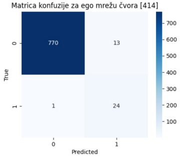
- Matrice konfuzija za svaku ego mrežu

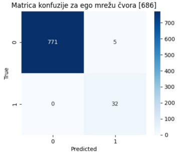


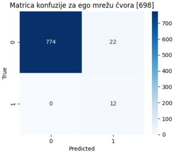


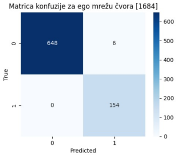


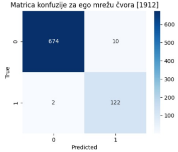
Kod i analiza - GCN

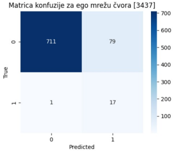


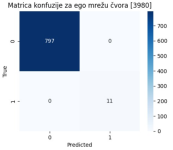


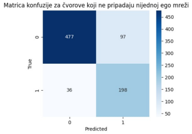












GAT

- Graph Attention Networks - arhitektura neuronskih mreža zasnovanih na grafovima koja koristi mehanizam pažnje za efikasnije učenje reprezentacija čvorova
  - za razliku od GCN-a, GAT omogućava da se različitim čvorovima u susedstvu pridaju različite važnosti tokom računanja agregata čvorova
  - GAT model koristi nešto što se zove glave pažnje (num heads) - svaka "glava" pažnje je kao dodatni sloj koji modelu omogućava da posmatra različite delove mreže iz različitih uglova
  - više glava pažnje omogućava modelu da nauči različite stvari o čvorovima u isto vreme

Kod i analiza - GAT

- Model

```
class GAT(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels, dropout_rate, num_heads=4):
        super(GAT, self).__init__()
        self.gat1 = GATConv(in_channels, hidden_channels, heads=num_heads, concat=True)
        self.dropout = torch.nn.Dropout(dropout_rate)
        self.gat2 = GATConv(hidden_channels * num_heads, out_channels, heads=1, concat=False)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.gat1(x, edge_index)
        x = F.elu(x)
        x = self.dropout(x)
        x = self.gat2(x, edge_index)
        return x
```

- Podešavanje hiperparametara

```
# Hiperparametri
hyperparameters = {
    'hidden_channels': [32, 64],
    'dropout_rate': [0.2, 0.5],
    'num_heads': [2, 4, 8],
    'num_epochs': [30, 60]
}
```

Kod i analiza - GAT

- Nakon treniranja modela za različite kombinacije hiperparametara i evaluacije na validacionom skupom, dobijeni su sledeci rezultati

```
Testing parameters: (32, 0.2, 2, 30)
[Train] Epoch: 0, Loss: 1.3278, Accuracy: 0.4525, Precision: 0.0522, Recall: 0.3926, F1: 0.0828
[Train] Epoch: 10, Loss: 0.2637, Accuracy: 0.9369, Precision: 0.6108, Recall: 0.9761, F1: 0.7159
[Train] Epoch: 20, Loss: 0.1992, Accuracy: 0.9483, Precision: 0.6681, Recall: 0.9814, F1: 0.7566
[Validate] Loss: 0.2099, Accuracy: 0.9493, Precision: 0.6773, Recall: 0.9758, F1: 0.7625
Testing parameters: (32, 0.2, 2, 60)
[Train] Epoch: 0, Loss: 1.3184, Accuracy: 0.5033, Precision: 0.0989, Recall: 0.4547, F1: 0.1502
[Train] Epoch: 10, Loss: 0.2882, Accuracy: 0.9427, Precision: 0.6284, Recall: 0.9716, F1: 0.7285
[Train] Epoch: 20, Loss: 0.1987, Accuracy: 0.9506, Precision: 0.6764, Recall: 0.9756, F1: 0.7607
[Train] Epoch: 30, Loss: 0.1829, Accuracy: 0.9510, Precision: 0.6782, Recall: 0.9807, F1: 0.7666
[Train] Epoch: 40, Loss: 0.1757, Accuracy: 0.9547, Precision: 0.7019, Recall: 0.9808, F1: 0.7848
[Train] Epoch: 50, Loss: 0.1723, Accuracy: 0.9554, Precision: 0.7176, Recall: 0.9819, F1: 0.7971
[Validate] Loss: 0.1866, Accuracy: 0.9529, Precision: 0.7259, Recall: 0.9776, F1: 0.8050
Testing parameters: (32, 0.2, 4, 30)
[Train] Epoch: 0, Loss: 1.3265, Accuracy: 0.5539, Precision: 0.0847, Recall: 0.4595, F1: 0.1300
[Train] Epoch: 10, Loss: 0.2365, Accuracy: 0.9365, Precision: 0.6308, Recall: 0.9806, F1: 0.7291
[Train] Epoch: 20, Loss: 0.1912, Accuracy: 0.9502, Precision: 0.6777, Recall: 0.9809, F1: 0.7673
[Validate] Loss: 0.1924, Accuracy: 0.9514, Precision: 0.6980, Recall: 0.9778, F1: 0.7818
```

- Najbolji model je model sa hiperparametrima

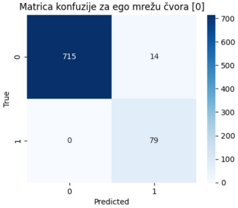
```
Best parameters: (32, 0.2, 8, 60), with loss: 0.1851
```

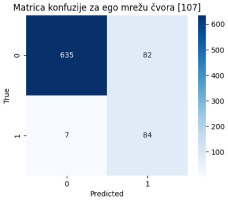
Kod i analiza - GAT

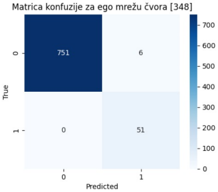
- Metrike na test skupu

```
Test Loss: 0.1698
Test Accuracy: 0.9597
Test Precision: 0.7767
Test Recall: 0.9756
Test F1 Score: 0.8428
```

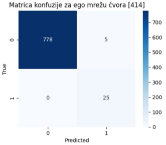
- Matrice konfuzija za svaku ego mrežu

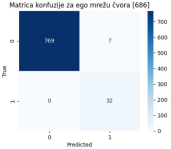


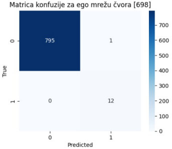


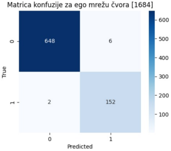


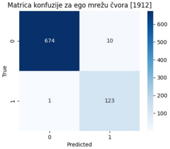
Kod i analiza - GAT

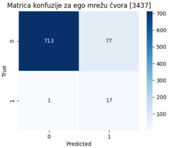


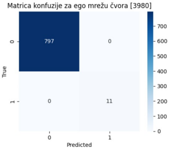


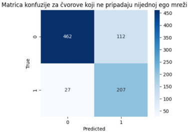












Poređenje modela

```
Test Loss: 0.1904
Test Accuracy: 0.9557
Test Precision: 0.7121
Test Recall: 0.9727
Test F1 Score: 0.7896
```

Slika: [GCN]

```
Test Loss: 0.1698
Test Accuracy: 0.9597
Test Precision: 0.7767
Test Recall: 0.9756
Test F1 Score: 0.8428
```

Slika: [GATs]

- Primetili smo da su modeli jako slični i da GAT ima blagu prednost, ali treniranje GCN modela je znatno brže, pa ih treba koristiti u skladu sa situacijom