Univerzitet u Beogradu
Matematički fakultet

GRAFOVSKE NEURONSKE MREŽE

Seminarski rad iz predmeta Računarska Inteligencija

Profesor:
Vladimir Filipović

Studenti:
Jelisaveta Gavrilović 188/2020
Marko Paunović 104/2020

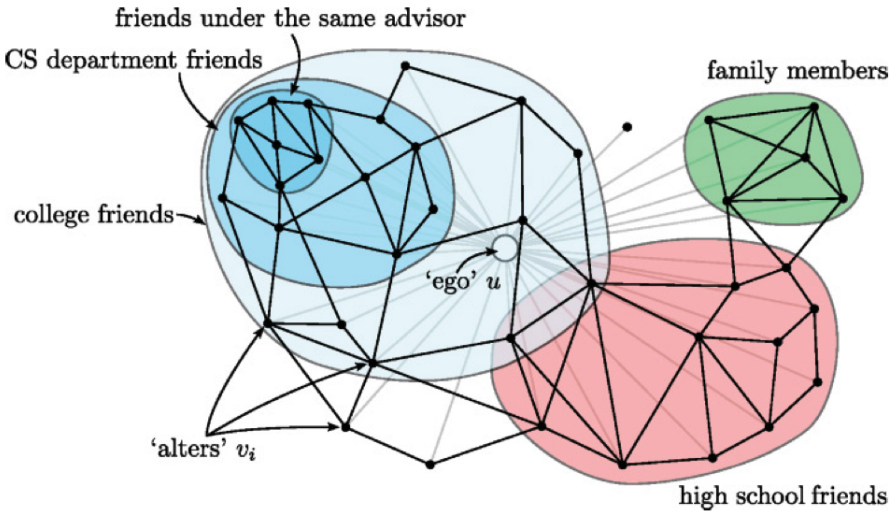
Beograd, septembar 2024.

Sadržaj

1 Uvod

Ego mreže su 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.



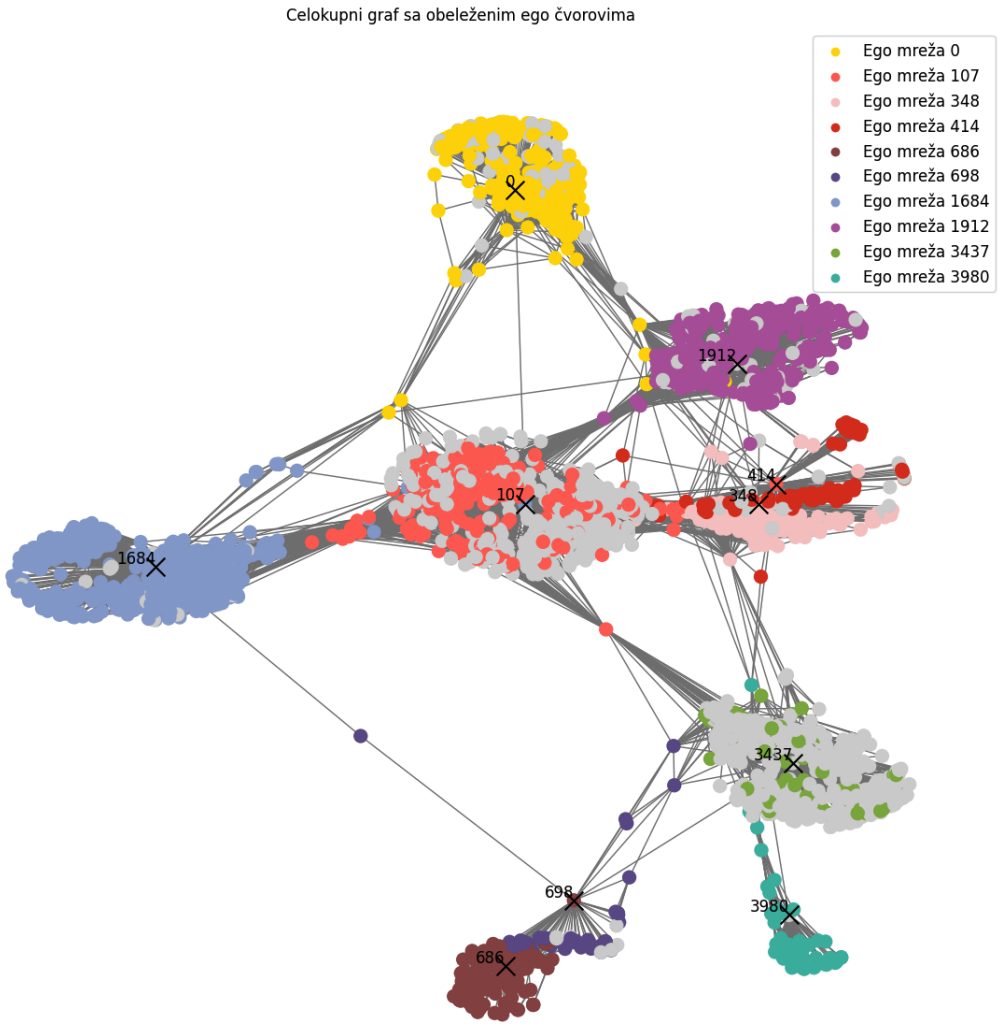
Slika 1: Primer ego mreže

2 Skup podataka

Skup podataka je preuzet sa sajta Stanford Network Analysis Project (SNAP). SNAP baza podataka je skup velikih društvenih i informacionih mreža koje se koriste za istraživanja u oblastima poput društvene analize, otkrivanja zajednica, rangiranja čvorova...

Mi smo se opredelili pecifično za Facebook podatke.

- Skup podataka je predstavljen u obliku grafa, gde:
  - čvorovi predstavljaju korisnike (ukupno 4039)
  - grane predstavljaju veze ili prijateljstva između dva korisnika (ukupno 88234)
- Ovaj graf je **neusmeren graf**, što znači da ako je korisnik A prijatelj korisnika B, onda je i korisnik B prijatelj korisnika A.
- Svaki korisnik ima i **niz atributa** koji opisuju njegove osobine i karakteristike (ukupno 2255).
- Međutim, pošto je skup podataka skinut sa Facebook-a, svi atributi su anonimizovani Facebook-ovom internom reprezentacijom. To znači da možemo samo da uporedimo da li korisnici imaju neke iste karakteristike, ali ne i šta konkretno one predstavljaju. Takođe, svi atributi su binarni, zapisani su u retkoj matrici koristeci "one hot encoding".



Slika 2: Struktura mreže sa obeleženim ego korisnicima i bojama ya različite ego mreže

3 Opis problema

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.

4 Opis rešenja

- 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 u raznim oblastima, uključujući biološke mreže, društvene mreže, finansije...
- 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.

- Graph Attention Networks - takođe specifičan tip GNN-a koji koristi mehanizam pažnje
  - mehanizam pažnje izračunava težine između čvora i njegovih suseda na osnovu njihovih karakteristika.
  - nakon toga dodeljuje različite težine susednim čvorovima tokom propagacije informacija kroz mrežu.

5 Kod i analiza

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

5.1 Učitavanje i priprema podataka

- Podatke smo učitali pomoću pickle biblioteke
- Čvorove smo podelili na trening, test i validacioni skup korišćenjem maski ya svaki od skupova

```
# 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))])
```

Slika 3: Struktura mreže sa obeleženim ego korisnicima i bojama za različite ego mreže

- Klase smo balansirali dodavajući im težine tako da ukoliko je klasa brojnija težina će biti manja i obrnuto.

5.2 Definicija modela

Napravili smo 2 modela:

- GCN - Graph Convolutional Networks

```python
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
```

Slika 4: GCN model

  – sadrži 2 konvoluciona sloja i dropout sloj između njih

- GAT - Graph Attention Networks

```python
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
```

Slika 5: GAT model

  – sadrži 2 konvoluciona sloja i dropout sloj između njih

5.3 Proces treniranja i evaluacije

- Pratili smo sledeće metrike:
  - Accuracy (tačnost)
  - Precision (preciznost)
  - Recall (odziv)
  - F1 score (F1 meru)
- kako bismo ocenili kvalitet modela i odlučili koji je bolji.

Koristeći validacioni skup proverom GCN modela za sledeće hiperparametre došli smo do onih parametara za koje se model najbolje ponaša.

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

Slika 6: Hiperparametri

```
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
```

Slika 7: Proces biranja najboljih parametara

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

Slika 8: Najbolji hiperparametri ya GCN model

Isto smo uradili i za GAT:

```python
# Hiperparametri
hyperparameters = {
    'hidden_channels': [32, 64],
    'dropout_rate': [0.2, 0.5],
    'num_heads': [2, 4, 8],
    'num_epochs': [30, 60]
}
learning_rate = 0.01
weight_decay = 5e-4
```

Slika 9: Hiperparametri

```
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
```

Slika 10: Proces biranja najboljih parametara

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

Slika 11: Najbolji hiperparametri ya GCN model

6 Poredjenje modela

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

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

Slika 12: Rezultati testiranja **GCN**

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

Slika 13: Rezultati testiranja **GAT**