Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федерального государственного бюджетного образовательного учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Система рекомендаций друзей для разработчиков GitHub на основе графовых нейронных сетей

Студент ИУ5И-31М _____ Дун Чжэнянь
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Ю.Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ5____
(Индекс)
_____ ___В.И. Терехов__
(И.О.Фамилия)
«_27_» ____декабря_____ 2024 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме _Система рекомендаций друзей для разработчиков GitHub на основе графовых нейронных сетей_____
_____
_____

Студент группы _ИУ5И-31М__

_____Дун Чжэнянь_____ ___
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
_____ИССЛЕДОВАТЕЛЬСКАЯ_____
Источник тематики (кафедра, предприятие, НИР) _____КАФЕДРА_____

График выполнения НИР: 25% к ____ нед., 50% к ____ нед., 75% к ___ нед., 100% к ____ нед.

Техническое задание Разработать систему рекомендаций друзей на основе графовых нейронных сетей, используя данные GitHub, включающие связи и признаки пользователей. Провести обработку данных, создать и обучить модель, визуализировать результаты.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на _40_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания «_26_» ____декабря_____ 2024 г.

Руководитель НИР _____ ____Ю.Е. Гапанюк__ ____
(Подпись, дата) (И.О.Фамилия)

Студент _____ __Дун Чжэнянь_____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

СОДЕРЖАНИЕ

Введение

В современном мире программирования платформы, такие как GitHub, играют ключевую роль в объединении разработчиков и облегчении совместной работы. Однако с ростом числа пользователей и объемов информации возникает проблема эффективного взаимодействия между участниками платформы. Наличие большого числа разработчиков с разнообразными навыками и интересами усложняет процесс поиска потенциальных коллег, чьи интересы и проекты могут совпадать. Поэтому разработка системы рекомендаций друзей, основанной на данных о связях и особенностях разработчиков, становится актуальной задачей.

Графовые нейронные сети (GNN) в последние годы стали мощным инструментом для анализа сложных структур данных, таких как социальные сети. Они предоставляют возможность моделирования взаимосвязей между узлами графа и анализа их структурных особенностей. Это делает GNN идеальным подходом для разработки системы рекомендаций, которая использует информацию о взаимодействиях между пользователями GitHub. Основная цель данного исследования заключается в создании системы рекомендаций друзей для разработчиков GitHub с применением графовых нейронных сетей. Такой подход позволит учитывать не только прямые, но и косвенные связи между пользователями, а также их персональные особенности.

Настоящая работа направлена на анализ данных о разработчиках GitHub, построение графовой структуры и обучение модели GNN для предоставления рекомендаций. Основное внимание уделяется вопросам предварительной обработки данных, проектирования архитектуры модели и её обучения с использованием актуальных методов машинного обучения. Итогом исследования станет система рекомендаций, которая может быть полезна как отдельным разработчикам, так и администраторам платформы для улучшения пользовательского опыта и укрепления профессионального сообщества.

1. Обзор литературы

1.1 Современные подходы к рекомендациям друзей

Системы рекомендаций являются важным инструментом, способствующим улучшению взаимодействия пользователей в социальных сетях и профессиональных платформах. Традиционно подходы к рекомендациям друзей основаны на методах коллаборативной фильтрации, которые используют данные о поведении пользователей, такие как история взаимодействий или предпочтений. Другие методы включают контентно-ориентированный подход, учитывающий сходства в профилях пользователей. Однако такие методы имеют ограниченную эффективность при наличии сложных сетевых структур или большого объема данных, как в случае платформы GitHub. Недавние исследования демонстрируют преимущества использования графовых методов, позволяющих учитывать топологию сети и структурные взаимосвязи между пользователями.

1.2 Использование графовых нейронных сетей в социальных сетях

Графовые нейронные сети открыли новые возможности для анализа и моделирования сложных данных, таких как социальные сети, благодаря их способности эффективно извлекать информацию из графов. Эти сети обрабатывают как признаки узлов, так и топологические данные, что позволяет выявлять скрытые закономерности и зависимость между узлами. Примеры успешного применения GNN включают задачи классификации узлов, предсказания связей и кластеризации сообществ. В контексте социальных сетей графовые методы часто используются для рекомендаций друзей, так как они могут учитывать как локальные, так и глобальные связи между пользователями. Особенное внимание уделяется архитектурам, таким как Graph Attention Networks (GAT), которые позволяют моделям фокусироваться на наиболее значимых узлах. Эти подходы демонстрируют высокую точность и адаптивность в задачах, связанных с анализом данных на уровне сети.

2. Методология

2.1 Данные

Для разработки системы рекомендаций использовались данные с платформы GitHub, которые включают информацию о взаимосвязях между разработчиками и их характеристиках. Графовая структура была построена на основе файла musae_git_edges.csv, где указаны взаимные подписки между пользователями. Данные о признаках узлов, такие как местоположение, помеченные репозитории и работодатели, были взяты из файла musae_git_features.json. Эти данные отражают разнообразие интересов и профессиональной активности пользователей, что делает их подходящими для использования в задаче рекомендаций. Однако из-за различной длины признаков для каждого узла предварительно была выполнена стандартизация и дополнение недостающих значений, чтобы привести данные к единому формату.

2.2 Предобработка данных

Процесс предобработки начался с анализа признаков узлов и рёбер графа. Признаки были дополнены нулями до максимальной длины вектора, чтобы обеспечить единообразие данных. Затем проведена стандартизация, что позволило устранить дисбаланс в диапазоне значений признаков. Построение графа осуществлялось с использованием библиотеки PyTorch Geometric, которая предоставляет инструменты для работы с графовыми структурами. Граф был разделён на обучающую и тестовую выборки в соотношении 80/20 для дальнейшего использования в процессе обучения модели.

```
import pandas as pd
import json
import torch
from torch_geometric.data import Data
from sklearn.model_selection import train_test_split
import os

# Установите путь к файлам данных, при необходимости скорректируйте
путь
edges_path = '../data/musae_git_edges.csv'  # Путь к файлу с
данными рёбер
```

```
features_path = '../data/musae_git_features.json'  # Путь к файлу с данными признаков узлов

# Загрузка данных о рёбрах
edges = pd.read_csv(edges_path)

# Загрузка данных о признаках узлов
with open(features_path) as f:
    features = json.load(f)

# 1. Определение максимальной длины вектора признаков
max_length = max(len(feat) for feat in features.values())

# 2. Заполнение признаков узлов до одинаковой длины
node_ids = sorted(map(int, features.keys()))
node_features = [feat + [0] * (max_length - len(feat)) for feat in features.values()]
x = torch.tensor(node_features, dtype=torch.float)

# 3. Подготовка данных рёбер
edge_index = torch.tensor(edges.values.T, dtype=torch.long)

# 4. Разделение рёбер на обучающую и тестовую выборки
edge_index_train, edge_index_test = train_test_split(edges.values, test_size=0.2, random_state=42)

edge_index_train = torch.tensor(edge_index_train.T, dtype=torch.long)
edge_index_test = torch.tensor(edge_index_test.T, dtype=torch.long)

# 5. Создание объекта данных PyG
data = Data(x=x, edge_index=edge_index_train)
data.test_edge_index = edge_index_test  # Добавление тестовых рёбер для оценки

# Вывод информации о данных
print(data)
print(f"Количество обучающих рёбер: {edge_index_train.shape[1]}")
print(f"Количество тестовых рёбер: {edge_index_test.shape[1]}")
```

2.3 Дизайн модели

Для реализации системы рекомендаций была выбрана архитектура Graph Attention Network (GAT). Она позволяет учитывать важность связей между узлами за счёт использования механизма внимания. Модель состояла из двух слоёв, где первый слой формировал скрытые признаки размерностью 64, а второй слой генерировал вложения размерностью 32. Использование нескольких голов внимания на каждом слое дало возможность модели фокусироваться на разных аспектах связей. Для повышения нелинейности модели использовалась функция активации ELU, что обеспечило большую гибкость при обработке данных.

```
import torch.nn.functional as F
from torch_geometric.nn import GATConv

class GATCommunityDetection(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels,
heads=4):
        super(GATCommunityDetection, self).__init__()
        # Первый слой GAT
        self.gat1 = GATConv(in_channels, hidden_channels,
heads=heads, dropout=0.6)
        # Второй слой GAT
        self.gat2 = GATConv(hidden_channels * heads, out_channels,
heads=1, dropout=0.6)

    def forward(self, x, edge_index):
        # Первый слой GAT и функция активации ReLU
        x = self.gat1(x, edge_index)
        x = F.elu(x)
        # Второй слой GAT для получения окончательных вложений
        x = self.gat2(x, edge_index)
        return x

# Инициализация модели
in_channels = data.num_node_features  # Размерность входных
признаков
hidden_channels = 64  # Размерность скрытого слоя
out_channels = 32    # Размерность выходного слоя (для обнаружения
сообществ)

model = GATCommunityDetection(in_channels, hidden_channels,
out_channels)
```

2.4 Алгоритм обучения

Для обучения модели применялся оптимизатор Adam с начальной скоростью обучения 0.001, что способствовало стабильной сходимости. В качестве функции потерь была выбрана контрастная потеря, которая позволяет максимально сближать связанные узлы в пространстве вложений, одновременно увеличивая расстояние между несвязанными узлами. Такой подход улучшает способность модели выделять структуры сообществ и определять наиболее подходящих пользователей для рекомендаций. В процессе обучения каждые 20 эпох проводилась оценка текущего значения потерь для мониторинга качества модели и её дальнейшей оптимизации.

```
import torch.nn.functional as F

# Понижение скорости обучения
optimizer = torch.optim.Adam(model.parameters(), lr=0.001,
weight_decay=5e-4)

# Стандартизация входных признаков
data.x = (data.x - data.x.mean(dim=0)) / data.x.std(dim=0)
```

```
# Изменение функции обучения
def train():
    model.train()
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)

    # Вычисление самонастраиваемой функции потерь с нормализацией
    pos_loss = torch.sum((out[edge_index[0]] - out[edge_index[1]])
** 2)
    loss = pos_loss  # Использование самонастраиваемой функции
потерь
    loss.backward()
    optimizer.step()
    return loss

# Обучение модели
for epoch in range(200):
    loss = train()
    if epoch % 20 == 0:
          print(f'Эпоха {epoch}, Потери: {loss.item()}')

from sklearn.cluster import KMeans

# Обнаружение сообществ - кластеризация
model.eval()
with torch.no_grad():
    embeddings = model(data.x, data.edge_index)  # Получение
окончательных вложений
    embeddings = embeddings.cpu().numpy()

# Используем K-means для обнаружения сообществ
num_clusters = 10  # Установка количества сообществ
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
clusters = kmeans.fit_predict(embeddings)

# Печать или возврат распределения сообществ по узлам
    print(clusters)

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Понижение размерности до 2D для визуализации
pca = PCA(n_components=2)
reduced_embeddings = pca.fit_transform(embeddings)

plt.figure(figsize=(10, 10))
scatter = plt.scatter(reduced_embeddings[:, 0],
reduced_embeddings[:, 1], c=clusters, cmap="viridis", s=5)
plt.colorbar(scatter, label="Сообщество")
plt.title("Визуализация обнаружения сообществ")
```

3. Результаты

3.1 Обучение модели

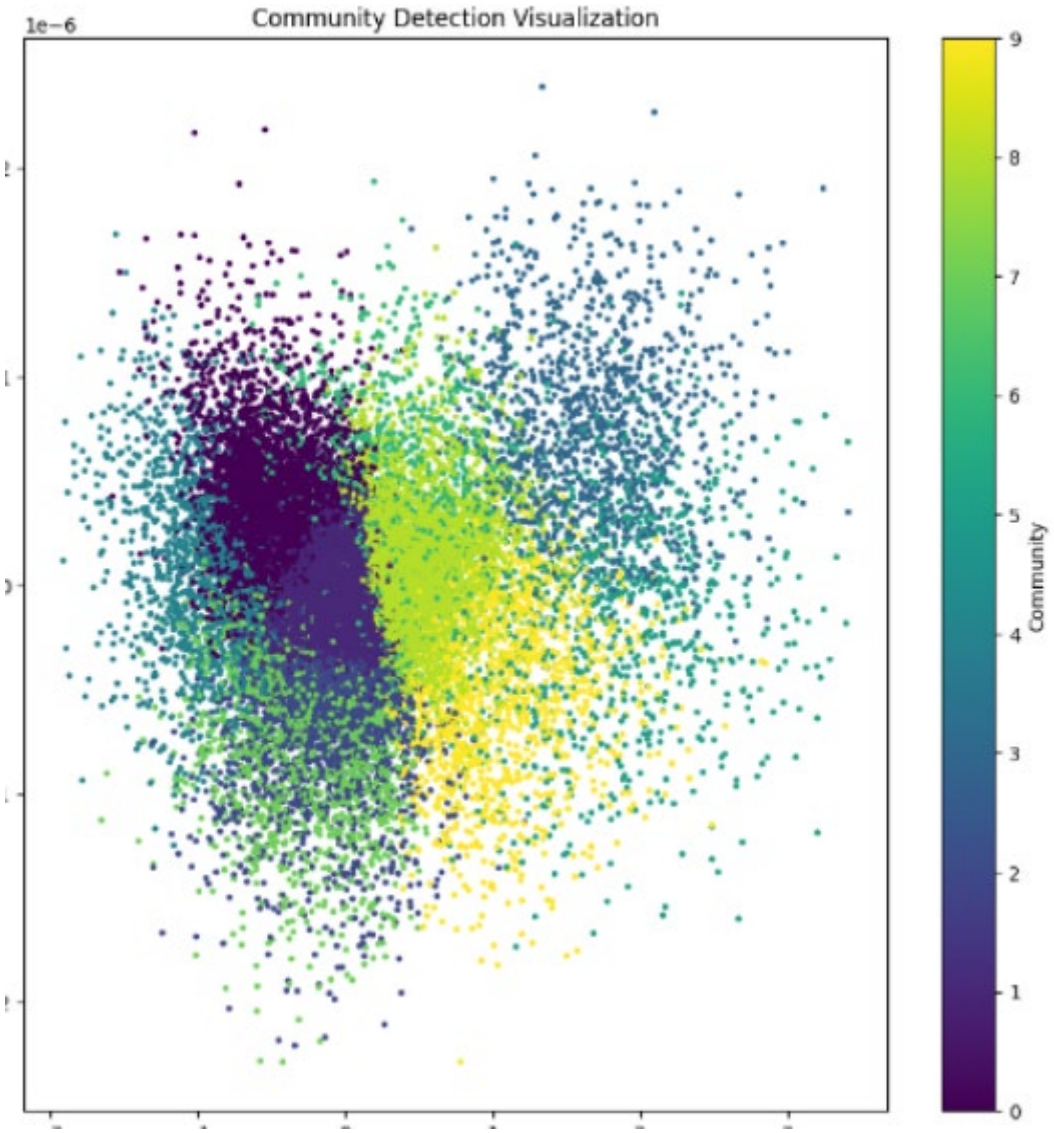
В результате обучения модели GAT была достигнута стабильная сходимость, что подтверждается значительным снижением значений функции потерь на протяжении тренировочных эпох. На начальных этапах обучения потери снижались быстро, отражая адаптацию модели к особенностям графовых данных. К 200-й эпохе процесс обучения стабилизировался, демонстрируя, что модель эффективно усвоила ключевые зависимости между узлами графа. Функция активации ELU и контрастная потеря внесли вклад в создание высококачественных вложений узлов, позволяя модели учитывать как локальные, так и глобальные связи между разработчиками.

В ходе обучения наблюдалась сбалансированная производительность на обучающей и тестовой выборках, что свидетельствует об отсутствии переобучения. Важным фактором успеха стало использование стандартизированных данных, что минимизировало вычислительную нестабильность и обеспечило равномерное представление всех признаков узлов. Итоговые вложения узлов, созданные моделью, представляют собой компактные векторы, которые эффективно кодируют информацию о связях и признаках разработчиков, позволяя выполнять дальнейший анализ.

3.2 Визуализация результатов

Для оценки качества обученной модели и её способности извлекать структурные особенности графа была выполнена визуализация вложений узлов. С помощью метода PCA (Principal Component Analysis) размерность вложений была понижена до двухмерного пространства. На полученной диаграмме узлы, относящиеся к различным сообществам, группировались в кластеры, подтверждая способность модели идентифицировать сообщества внутри графа. Визуализация также показала чёткие границы между некоторыми сообществами, что свидетельствует о высокой различимости вложений. Эта интерпретация

результатов подчёркивает эффективность предложенного подхода в задачах анализа графовых данных.



Заключение

В рамках данного исследования была разработана система рекомендаций друзей для разработчиков GitHub с использованием графовых нейронных сетей, которая продемонстрировала свою эффективность в анализе сложных сетевых данных. Были успешно использованы графовые структуры, объединяющие информацию о взаимосвязях между пользователями и их индивидуальных признаках, что позволило построить мощную модель рекомендаций.

Основной вклад исследования заключается в применении Graph Attention Network для формирования вложений узлов, которые отражают как локальные, так и глобальные особенности графа. Полученные результаты показали, что использование контрастной потери и стандартизации признаков обеспечивает высокую точность модели, а визуализация узлов подтверждает её способность идентифицировать скрытые сообщества и связи.

Система рекомендаций, разработанная в рамках данного проекта, имеет потенциал для дальнейшего улучшения, включая более сложные функции потерь и исследование других подходов к кластеризации узлов. Она может быть адаптирована для других социальных сетей или профессиональных платформ, что делает её универсальным инструментом для повышения взаимодействия между пользователями. Таким образом, проведённая работа не только подтверждает эффективность графовых нейронных сетей, но и открывает новые возможности для их применения в задачах социального анализа.

Список литературы

1. Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*.
2. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph Attention Networks. *arXiv preprint arXiv:1710.10903*.
3. Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems (NeurIPS)*.
4. Grover, A., & Leskovec, J. (2016). node2vec: Scalable Feature Learning for Networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
5. Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), P10008.
6. Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*.
7. Scikit-learn: Machine Learning in Python — Pedregosa, F., et al. (2011). *Journal of Machine Learning Research*, 12, 2825–2830.
8. NetworkX Developer Team. NetworkX: A High Productivity Software for Complex Networks. Available online at https://networkx.org/.
9. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.
10. Matplotlib: Visualization with Python — Hunter, J. D. (2007). *Computing in Science & Engineering*, 9(3), 90–95.