NATIONAL RESEARCH UNIVERSITY HIGHER SCHOOL OF ECONOMICS
GRADUATE SCHOOL OF BUSINESS

SOCIAL NETWORK ANALYSIS
Comparison of the results of clustering of VK groups and classification using machine learning methods

Students of group BA&BigData232
Illarionov Alexey
Bolotova Ekaterina

Moscow, 2024

1. Note

Previously, the work had the topic "Analysis of the user's city and university based on the VK groups to which the user is subscribed", but as part of the course we decided to simplify the topic to the current one.

2. Data

In the current work, we have taken the VK of the group as an object of analysis. At the beginning of the work, we hypothesized that it is possible to determine whether a group belongs to a certain classification, in our case it is a university, based on the connections of groups with the main representative groups of universities.

To carry out the data collection stage about VK groups, we used the VK developer API, for this we created an application in VK and wrote code in Java Spring to upload data. We have selected 16 universities with an open database of group members in VK.

Groups were uploaded according to 16 universities. The next step was to upload information about university users in proportion to the number of users in the group (that is, if the group initially had 20,000 users, then we uploaded only 2000 participants to speed up data collection). And in the last step, we uploaded data for all the groups of participants that we uploaded in the previous step in order to find potentially related groups with official university associations.

Next, we cleared the database of garbage groups, where the number of users was disproportionately small, eventually leaving about 4,000 groups. Next, we created a table that contains information about the number of identical participants between the two groups, thereby forming a list of graph edges.

The structure of nodes:

1. Group Id;
2. Group name;

3. University id;
4. Is the group the main one;
5. Number of group members;

The structure of edges:

1. Group Id №1,
2. Group Id №2,
3. The number of identical participants.

Figure 1 below shows a graph built using the networkx library

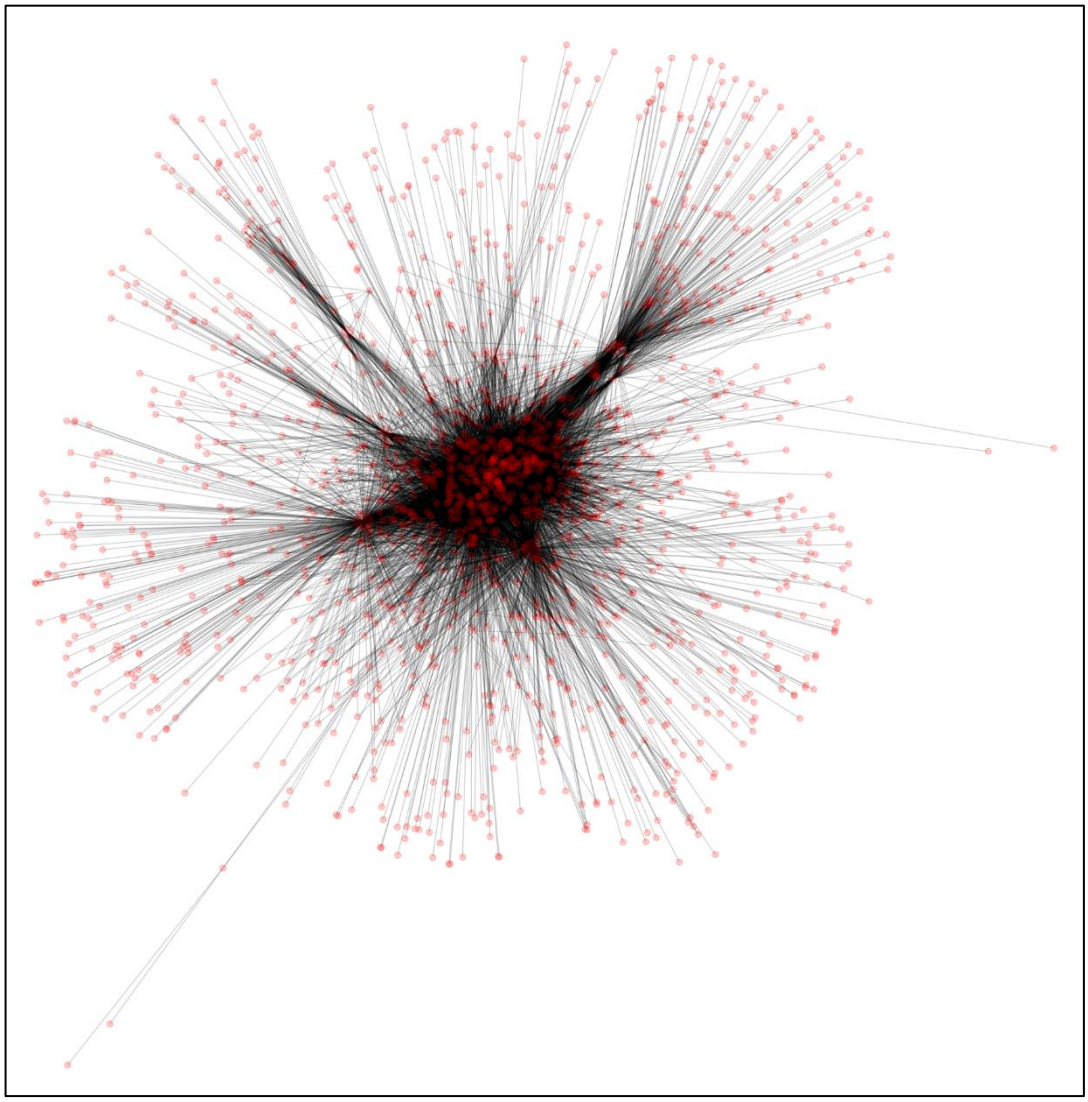

Figure 1. Initial graph(node’s size depends on the number of group members)

3. Community detection analysis

First of all, we decided to check how applicable clustering is here, as a method of analyzing groups. As part of the current task, we used the Louvain method. Using the community library, we wrote the code below

```
G = nx.Graph(directed = False)
min_q = sys.maxsize
max_q = 0

with open('dataInfo/nodes.csv', newline='', encoding='utf-8') as csvfile:
    csvreaderNode = csv.reader(csvfile, delimiter=";")
    next(csvreaderNode)
    for row in csvreaderNode:
         G.add_node(int(row[0]), name=row[2], quantity=int(row[3]))
         min_q = min(min_q, int(row[3]))
         max_q = max(max_q, int(row[3]))

with open('dataInfo/edges.csv', newline='', encoding='utf-8') as csvfile:
    csvreaderEdge = csv.reader(csvfile, delimiter=";")
    next(csvreaderEdge)
    for row in csvreaderEdge:
        G.add_edge(int(row[0]), int(row[1]), weight=int(row[2]))

partition = community_louvain.best_partition(G, weight="weight")

for node, community in partition.items():
    G.nodes[node]['community'] = community

pos = nx.spring_layout(G)

colors = [partition[node] for node in G.nodes()]
sizes =  [normalize(G.nodes[node]['quantity'], min_q, max_q, 50, 500) for node in
G.nodes()]
alpha =  [normalize(G.nodes[node]['quantity'], min_q, max_q, 0.5, 1) for node in
G.nodes()]

print("Community quantity: "+str(len(set(partition.values()))))

plt.figure(figsize=(12, 12))
nx.draw_networkx_edges(G, pos, alpha=0.5)
nx.draw_networkx_nodes(G, pos, node_size=sizes, node_color=colors,
cmap=plt.cm.rainbow, alpha=alpha)
plt.title('Community Detection Visualization')
plt.show()
```

This code downloads information from a CSV file and generates a graph based on the data from the file (in this case, the code is not optimal, then we used the pandas library to improve the purity of the code). After that, it forms a community and fixes it as a node parameter. After that, the color is calculated based on the community, and then the size and alpha of the node are calculated based on the number of participants. After that, the graph is drawn.

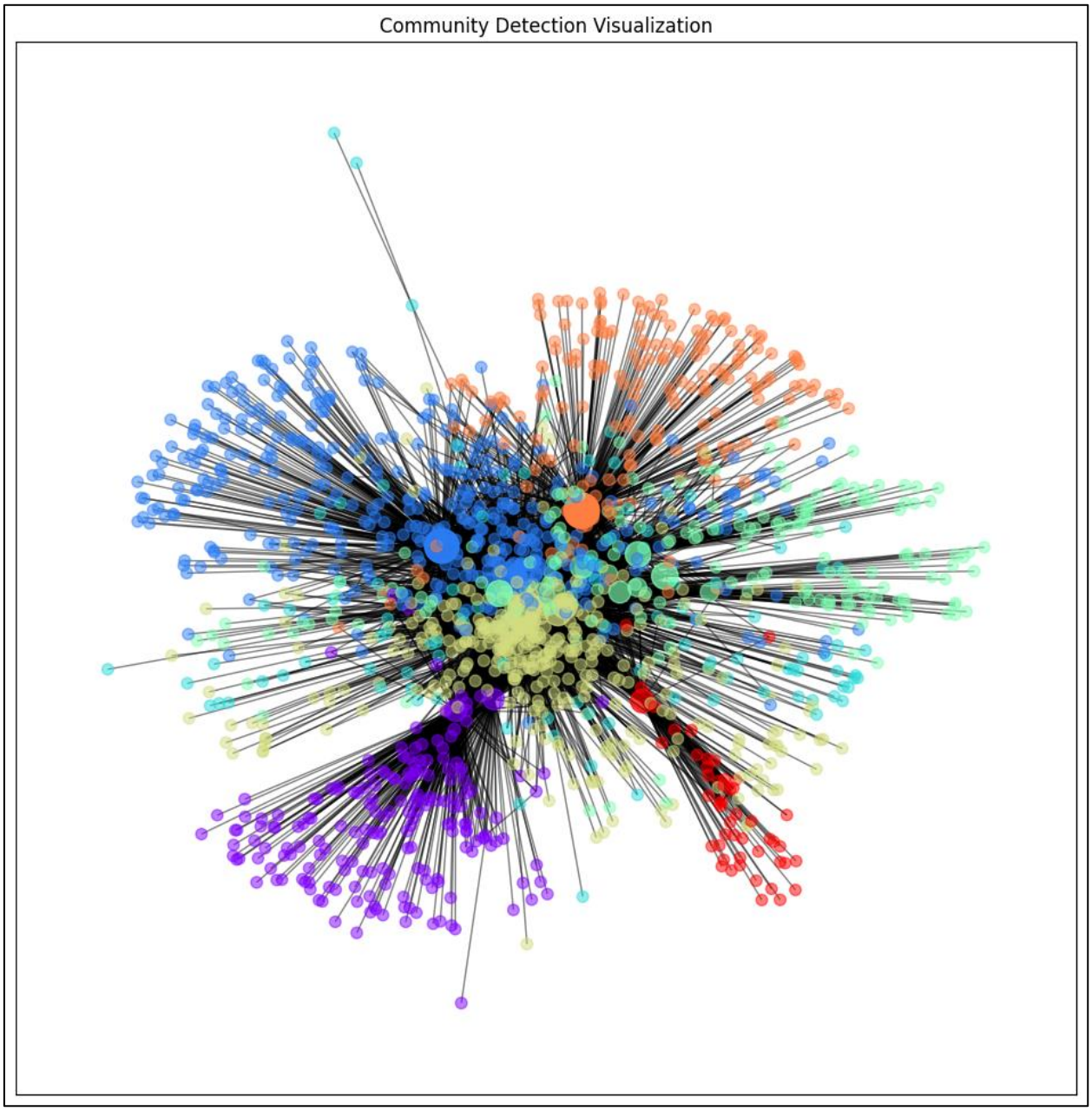

Figure 2. Community detection graph

As we can see, the algorithm identified only 7 communities, while we expected to see a number close to 16, moreover, we expected that there would be

groups that could not be attributed to a competitive university, so this algorithm is not applicable in the current case and it is necessary to resort to more complex methods of analysis.

4. Classifying groups by distribution using «Random Forest Classifier»

As part of our nodes, we prepared a test sample of groups of 400 elements, within which we were able to accurately identify universities or that the group does not belong to any university and also recorded information on the distribution within each university. Next, using pandas and sklearn, we wrote the code shown below.

```
nodes = pd.read_csv('dataInfo/nodes.csv', delimiter=';')
edges = pd.read_csv('dataInfo/edges.csv', delimiter=';')


perameters = ['sub_quantity','open_sub_quantity',
              'MGU_quantity','MIFI_quantity','HSE_quantity',
              'SECH_quantity','MEI_quantity','MGIMO_quantity',
              'PNIPU_quantity','MIREA_quantity','MTUSI_quantity',
              'MAI_quantity','PGNIU_quantity','MFTI_quantity',
              'RUDN_quantity','ITMO_quantity','REU_quantity','MGTU_quantity']
target = 'university'

university_filled = nodes[nodes[target].notna()]

X = university_filled[perameters]
y = university_filled[target]

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,
random_state=42)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_val)

accuracy = accuracy_score(y_val, y_pred)
print("Accuracy on validation set: {:.2f}%".format(accuracy * 100))

df_unfilled = nodes[nodes[target].isna()]
X_unfilled = df_unfilled[perameters]
nodes.loc[nodes[target].isna(), target] = model.predict(X_unfilled)
```

As a result, we got a good result with an accuracy of 90% on the validation data. Based on the software data, a graph was constructed, which is presented below.

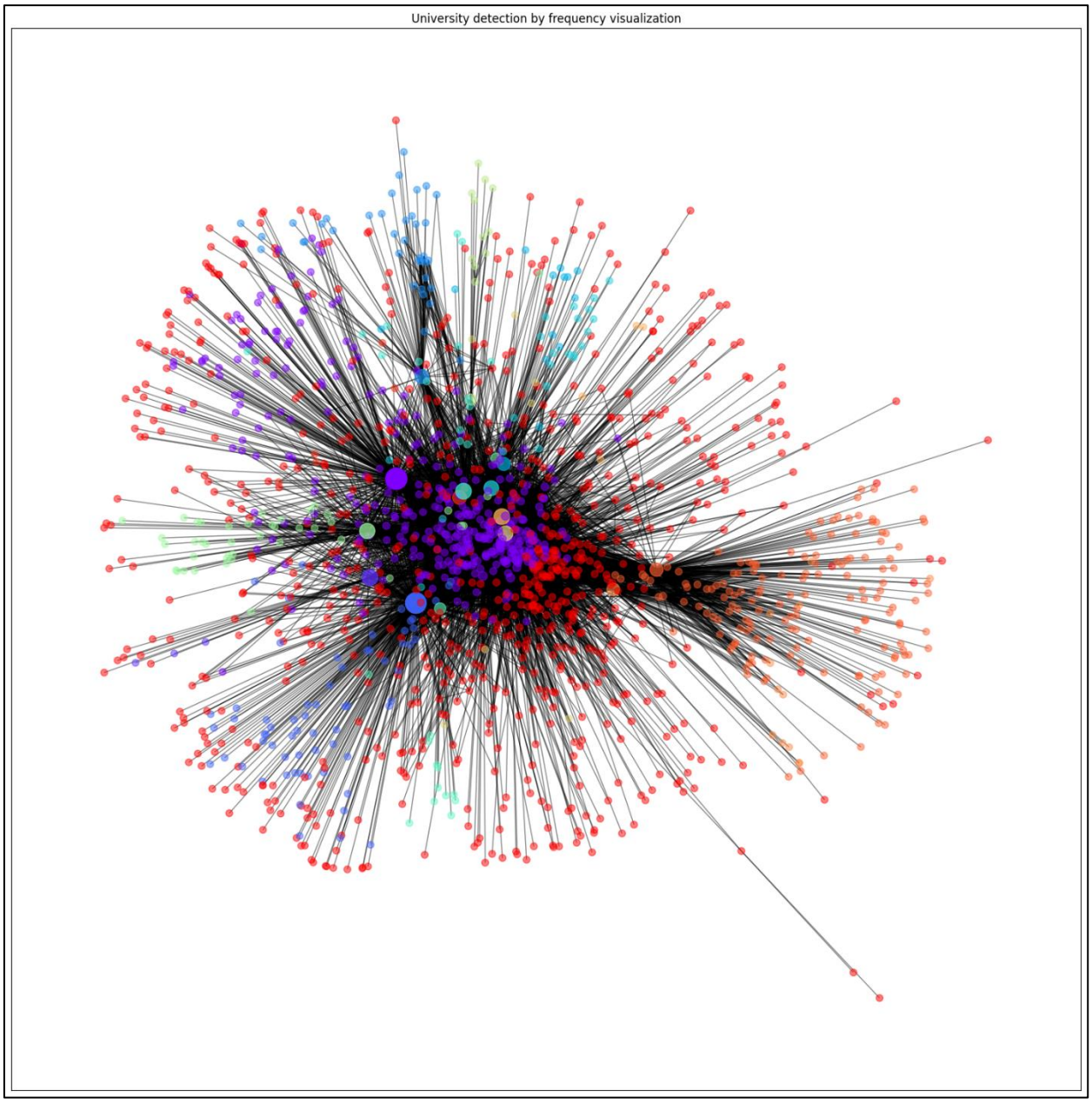

Figure 3. University detection by frequency graph

However, despite the accuracy, we can safely assume that the results obtained cannot be trusted (probably due to the small variety of test data), since the graph shows too many groups that belong to the same university, for example, to Moscow State University - purple elements.

5. Classifying groups by graph analysis

Using GCN and Torch libraries, we implemented code that takes a test graph as input, learns 200 epochs on it, and then runs the original graph on the model. The written code is presented below.

```
def build_graph(node_df, edge_df):

    G = nx.Graph()
    for _, row in node_df.iterrows():
        university = row['custom_university_id']
        if pd.isnull(row['custom_university_id']):
            university = np.nan
        else:
            university = int(row['custom_university_id'])

        G.add_node(row['custom_id'], name = row['name'], university=university,
is_main_group=int(row['is_main_group_int']),
sub_quantity=int(row['sub_quantity']))

    for _, row in edge_df.iterrows():
        if row['vk1_custom_id'] in G.nodes and row['vk2_custom_id'] in G.nodes:
            G.add_edge(row['vk1_custom_id'], row['vk2_custom_id'],
weight=int(row['Quantity']))

    return G

nodes = pd.read_csv("dataset/nodes.csv", delimiter= ';')
edges = pd.read_csv("dataset/edges.csv", delimiter= ';')
graph = build_graph(nodes, edges)


train_nodes = pd.read_csv("dataset/testnodes.csv", delimiter= ';')
train_edges = pd.read_csv("dataset/testedges.csv", delimiter= ';')
train_graph = build_graph(train_nodes, train_edges)

valid_nodes = pd.read_csv("dataset/validnodes.csv", delimiter= ';')
valid_edges = pd.read_csv("dataset/validedges.csv", delimiter= ';')
valid_graph = build_graph(valid_nodes, valid_edges)

def prepare_data(graph):
    node_features = []
    node_labels = []
    for node in graph.nodes(data=True):
        features = [node[1]['is_main_group'], node[1]['sub_quantity']]
        label = node[1]['university']
        node_features.append(features)
        node_labels.append(label)
    edge_index = torch.tensor(list(graph.edges)).t().contiguous()
```

```
    x = torch.tensor(node_features, dtype=torch.float)
    y = torch.tensor(node_labels, dtype=torch.long)
    return Data(x=x, edge_index=edge_index, y=y)

train_data = prepare_data(train_graph)
valid_data = prepare_data(valid_graph)

class GCN(torch.nn.Module):
    def __init__(self, num_node_features, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(num_node_features, 16)
        self.conv2 = GCNConv(16, num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

model = GCN(num_node_features=train_data.num_features,
num_classes=len(set(train_data.y.tolist())))
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()

def train():
    model.train()
    optimizer.zero_grad()
    out = model(train_data)
    loss = criterion(out[train_data.train_mask],
train_data.y[train_data.train_mask])
    loss.backward()
    optimizer.step()
    return loss.item()

# Проверка модели на валидационных данных
def test(data):
    model.eval()
    out = model(data)
    pred = out.argmax(dim=1)
    correct = pred[data.val_mask].eq(data.y[data.val_mask]).sum().item()
    acc = correct / data.val_mask.sum().item()
    return acc

# Метки для обучения и валидации
train_mask = torch.ones(train_data.num_nodes, dtype=torch.bool)
train_data.train_mask = train_mask
val_mask = torch.ones(valid_data.num_nodes, dtype=torch.bool)
valid_data.val_mask = val_mask

for epoch in range(200):
    loss = train()
```

```
    valid_acc = test(valid_data)
    print(f'Epoch {epoch}, Loss: {loss}, Validation Accuracy: {valid_acc}')

def fill_missing_values(graph, model):
    model.eval()
    node_features = []
    nodes = list(graph.nodes(data=True))
    for node in nodes:
        features = [node[1]['is_main_group'], node[1]['sub_quantity']]
        node_features.append(features)
    x = torch.tensor(node_features, dtype=torch.float)
    edge_index = torch.tensor(list(graph.edges)).t().contiguous()
    data = Data(x=x, edge_index=edge_index)
    out = model(data)
    pred = out.argmax(dim=1)
    for idx, node in enumerate(nodes):
        if pd.isna(node[1]['university']):
            node[1]['university'] = pred[idx].item()

fill_missing_values(graph, model)
```

In the end, we got an accuracy rate of just over 30%. and we got the following graph as a result. In the end, we got an accuracy rate of just over 30%. As a result, we have obtained the graph, which is presented below. In the end, although the accuracy of the definition is low, we got a picture more similar to the expected result, when most of the groups are not defined (marked in red), and there are only a small number of groups that actually belong to universities.

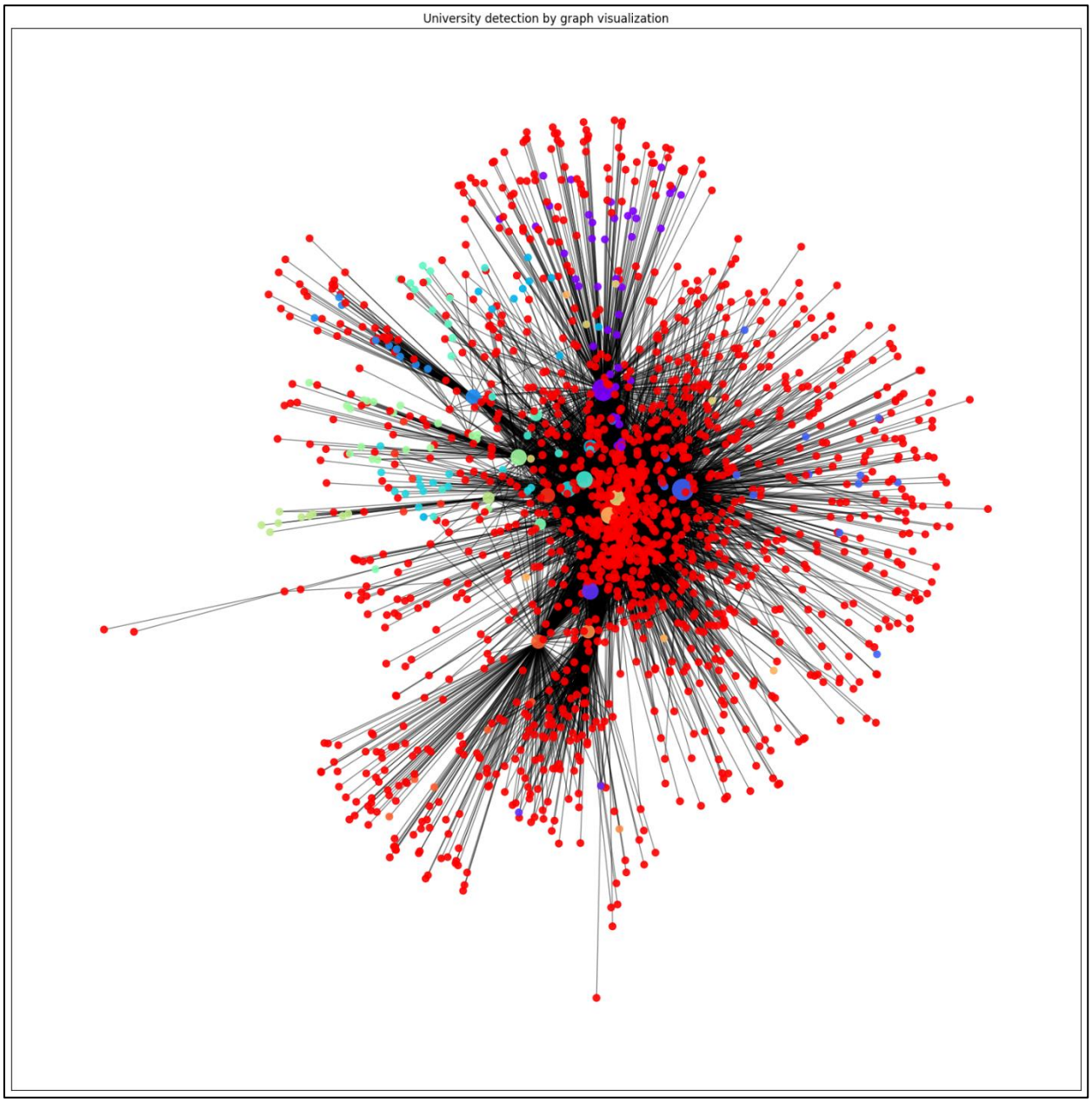

Figure 4. University detection by graph

6. Conclusion

As part of our project, we explored several data analysis techniques to put into practice the principles of half-point scoring within the course. The traditional method of community detection in this case was not successful, identifying too few communities, which was logical given the complexity of the original graph and classification criteria.

The student distribution analysis approach yielded good accuracy on the validation data, although it is worth noting that the results were not entirely as expected, likely due to the limited amount of training data.

Graph analysis methods showed lower accuracy in the validation test, but surprisingly, the final outcome was more in line with our expectations. However, a closer examination of the results suggested that the data provided to the model was insufficient and further refinement of the model would be necessary.

Therefore, we have left space for further analysis of this issue and the development of a more sophisticated method of analysis.