BONUS

Approach 1

I utilized a graph convolutional network (GCN) to analyze the dataset. GCNs are well suited for this task because they effectively take into account a node's features as well as a node's neighbors. Through very limited training, I was able to get impresive results with the GCN

GCN Results

Epoch: 250, Loss: 0.3857, Train Acc: 0.9600, Val Acc: 0.8504, Test Acc: 0.8480

Classification Report:

Node	precision	recall	f1-score	support
0	0.87	0.92	0.90	105
1	1.00	0.80	0.89	10
2	1.00	0.50	0.67	4
3	0.83	0.91	0.86	53
4	0.00	0.00	0.00	1
5	0.78	0.72	0.75	29
6	0.79	0.73	0.76	66
7	0.83	0.56	0.67	9
8	0.93	0.81	0.87	52
9	1.00	0.57	0.73	7
10	0.90	0.94	0.92	128
11	1.00	0.81	0.90	16
12	0.67	0.33	0.44	6
13	1.00	0.86	0.92	7
14	0.79	0.80	0.79	70
15	0.95	0.78	0.86	23
16	0.85	0.52	0.65	21
17	0.80	0.94	0.86	156
accuracy	_	_	0.85	763
macro_avg	0.83	0.69	0.75	763
weighted_avg	0.85	0.85	0.84	763

ROC AUC Score (One-vs-Rest): 0.96 (very high!!)

Approach 2

I also attempted to use simple logistic regression on this task. This trains a classifier on the matrix made up of node features.

Logistic Regression Results

Classification Report:

```
     Node      precision    recall  f1-score   support

        0       0.86      0.86      0.86       225
        1       0.00      0.00      0.00         6
        2       0.75      0.27      0.40        11
        3       0.71      0.75      0.73        96
        4       0.00      0.00      0.00         2
        5       0.75      0.74      0.75        78
        6       0.67      0.68      0.67       142
        7       0.80      0.17      0.29        23
        8       0.87      0.70      0.77        93
        9       1.00      0.11      0.20         9
       10       0.81      0.88      0.84       269
       11       0.80      0.70      0.74        23
       12       0.33      0.11      0.17         9
       13       1.00      0.44      0.61        16
       14       0.72      0.65      0.68       117
       15       0.85      0.63      0.72        46
       16       0.83      0.58      0.68        50
       17       0.71      0.90      0.79       310

accuracy                           0.76      1525
macro avg      0.69      0.51      0.55      1525
weighted avg   0.77      0.76      0.75      1525
```

One-vs-Rest ROC AUC Score: 0.92

Overall Results

I found that the GCN was more accurate than logistic regression for this task. GCN had an accuracy of 85% and logistic regression had an accuracy of 75%. I did not perform any hyperparameter tuning and I'm sure that the performance could be boosted even higher with a properly tuned GCN.

```
import os
import json
import pandas as pd
import torch
import torch.nn.functional as F
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv

from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score
```

```
from sklearn.preprocessing import label_binarize

import numpy as np
```

GCN

```
import os
import json
import pandas as pd
import numpy as np
import torch
import torch.nn.functional as F
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv

# ----------------------------
# 1. Load Data
# ----------------------------
features_path =
'/Users/log/Github/Spring2025Classes/social_networks/project4/lasftm_asia/
lastfm_asia_features.json'
targets_path =
'/Users/log/Github/Spring2025Classes/social_networks/project4/lasftm_asia/
lastfm_asia_target.csv'
edges_path   =
'/Users/log/Github/Spring2025Classes/social_networks/project4/lasftm_asia/
lastfm_asia_edges.csv'

# Load node features from JSON.
with open(features_path, 'r') as f:
    features_dict = json.load(f)

# Convert the features dictionary into a sorted list (assuming keys are
node IDs as strings).
node_ids = sorted(features_dict.keys(), key=lambda x: int(x))
features_list = [features_dict[node_id] for node_id in node_ids]

# ----------------------------
# 2. Create a Multi-hot Encoding for Features
# ----------------------------
# Build a set of all unique artist IDs
all_artists = set()
for feats in features_list:
    all_artists.update(feats)
all_artists = sorted(all_artists)
artist_to_index = {artist: i for i, artist in enumerate(all_artists)}

# Create a fixed-dimension feature matrix: (num_nodes, num_artists)
num_nodes = len(features_list)
```

```
num_artists = len(artist_to_index)
features_matrix = np.zeros((num_nodes, num_artists), dtype=np.float32)

for i, feats in enumerate(features_list):
    for artist in feats:
        features_matrix[i, artist_to_index[artist]] = 1

# Convert the numpy array to a torch tensor
x = torch.tensor(features_matrix, dtype=torch.float)

# ---------------------------
# 3. Load Targets and Edges
# ---------------------------
# Load targets. Assumes CSV has columns 'id' and 'target'.
targets_df = pd.read_csv(targets_path)
y = torch.tensor(targets_df['target'].values, dtype=torch.long)

# Load edges. Assumes CSV has columns 'node_1' and 'node_2'.
edges_df = pd.read_csv(edges_path)
edge_index = torch.tensor(edges_df[['node_1', 'node_2']].values.T,
dtype=torch.long)

# ---------------------------
# 4. Create the PyG Data Object and Data Splits
# ---------------------------
data = Data(x=x, edge_index=edge_index, y=y)

# Create boolean masks for train/validation/test splits (80/10/10)
train_mask = torch.zeros(data.num_nodes, dtype=torch.bool)
val_mask   = torch.zeros(data.num_nodes, dtype=torch.bool)
test_mask  = torch.zeros(data.num_nodes, dtype=torch.bool)

perm = torch.randperm(data.num_nodes)
train_idx = perm[:int(0.8 * data.num_nodes)]
val_idx   = perm[int(0.8 * data.num_nodes):int(0.9 * data.num_nodes)]
test_idx  = perm[int(0.9 * data.num_nodes):]

train_mask[train_idx] = True
val_mask[val_idx] = True
test_mask[test_idx] = True

data.train_mask = train_mask
data.val_mask = val_mask
data.test_mask = test_mask

# ---------------------------
# 5. Define the GCN Model
# ---------------------------
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        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 = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

# --------------------------
# 6. Training Setup
# --------------------------
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GCN(in_channels=data.num_features,
            hidden_channels=16,
            out_channels=len(torch.unique(data.y))).to(device)
data = data.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-
4)

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

def evaluate(mask):
    model.eval()
    with torch.no_grad():
        out = model(data)
        pred = out.argmax(dim=1)
        correct = (pred[mask] == data.y[mask]).sum().item()
        acc = correct / int(mask.sum())
    return acc

# --------------------------
# 7. Train the Model
# --------------------------
for epoch in range(1, 251):
    loss = train()
    if epoch % 10 == 0:
        train_acc = evaluate(data.train_mask)
        val_acc = evaluate(data.val_mask)
        test_acc = evaluate(data.test_mask)
        print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}, '
              f'Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}, Test
Acc: {test_acc:.4f}')

# --------------------------
# 8. Extended Evaluation Metrics on the Test Set
# --------------------------
model.eval()
```

```
    with torch.no_grad():
        out = model(data)
        # Since our model outputs log probabilities, we convert them to
    probabilities
        prob_test = torch.exp(out[data.test_mask]).cpu().numpy()
        pred_test = np.argmax(prob_test, axis=1)
        true_test = data.y[data.test_mask].cpu().numpy()
```

```
Epoch: 010, Loss: 1.7449, Train Acc: 0.6531, Val Acc: 0.6312, Test Acc:
0.6566
Epoch: 020, Loss: 1.3136, Train Acc: 0.7685, Val Acc: 0.7415, Test Acc:
0.7523
Epoch: 030, Loss: 1.0830, Train Acc: 0.8131, Val Acc: 0.7769, Test Acc:
0.7759
Epoch: 040, Loss: 0.9443, Train Acc: 0.8478, Val Acc: 0.7979, Test Acc:
0.8126
Epoch: 050, Loss: 0.8422, Train Acc: 0.8710, Val Acc: 0.8071, Test Acc:
0.8152
Epoch: 060, Loss: 0.7702, Train Acc: 0.8852, Val Acc: 0.8163, Test Acc:
0.8126
Epoch: 070, Loss: 0.7010, Train Acc: 0.8952, Val Acc: 0.8215, Test Acc:
0.8152
Epoch: 080, Loss: 0.6540, Train Acc: 0.9051, Val Acc: 0.8215, Test Acc:
0.8178
Epoch: 090, Loss: 0.6307, Train Acc: 0.9097, Val Acc: 0.8215, Test Acc:
0.8178
Epoch: 100, Loss: 0.6074, Train Acc: 0.9139, Val Acc: 0.8281, Test Acc:
0.8244
Epoch: 110, Loss: 0.5989, Train Acc: 0.9175, Val Acc: 0.8333, Test Acc:
0.8349
Epoch: 120, Loss: 0.5527, Train Acc: 0.9228, Val Acc: 0.8333, Test Acc:
0.8322
Epoch: 130, Loss: 0.5377, Train Acc: 0.9274, Val Acc: 0.8373, Test Acc:
0.8336
Epoch: 140, Loss: 0.5311, Train Acc: 0.9320, Val Acc: 0.8386, Test Acc:
0.8401
Epoch: 150, Loss: 0.4957, Train Acc: 0.9367, Val Acc: 0.8438, Test Acc:
0.8414
Epoch: 160, Loss: 0.4675, Train Acc: 0.9377, Val Acc: 0.8465, Test Acc:
0.8440
Epoch: 170, Loss: 0.4611, Train Acc: 0.9424, Val Acc: 0.8438, Test Acc:
0.8414
Epoch: 180, Loss: 0.4435, Train Acc: 0.9454, Val Acc: 0.8504, Test Acc:
0.8453
Epoch: 190, Loss: 0.4272, Train Acc: 0.9505, Val Acc: 0.8491, Test Acc:
0.8388
Epoch: 200, Loss: 0.4083, Train Acc: 0.9513, Val Acc: 0.8543, Test Acc:
```

```
0.8414
Epoch: 210, Loss: 0.3904, Train Acc: 0.9551, Val Acc: 0.8543, Test Acc:
0.8388
Epoch: 220, Loss: 0.3995, Train Acc: 0.9587, Val Acc: 0.8530, Test Acc:
0.8427
Epoch: 230, Loss: 0.3779, Train Acc: 0.9572, Val Acc: 0.8530, Test Acc:
0.8467
Epoch: 240, Loss: 0.3730, Train Acc: 0.9598, Val Acc: 0.8517, Test Acc:
0.8453
Epoch: 250, Loss: 0.3857, Train Acc: 0.9600, Val Acc: 0.8504, Test Acc:
0.8480
```

```python
# Import additional metrics from scikit-learn

print("\n--- Extended Evaluation Metrics ---")
print("\nConfusion Matrix:")
print(confusion_matrix(true_test, pred_test))

print("\nClassification Report:")
print(classification_report(true_test, pred_test))

# For ROC AUC, we need to binarize the true labels.
num_classes = len(torch.unique(data.y))
true_test_binarized = label_binarize(true_test,
classes=list(range(num_classes)))

try:
    roc_auc = roc_auc_score(true_test_binarized, prob_test,
multi_class='ovr')
    print("\nROC AUC Score (One-vs-Rest):", roc_auc)
except Exception as e:
    print("\nCould not compute ROC AUC Score:", e)
```

```
--- Extended Evaluation Metrics ---

Confusion Matrix:
[[ 97   0   0   0   0   0   2   0   0   0   1   0   0   0   2   0   0   3]
 [  0   8   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   2   0   0   0   0   0   0   0   1   0   0   0   1   0   0   0]
 [  0   0   0  48   0   1   1   1   0   0   0   0   1   0   0   0   0   1]
 [  0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0]
 [  1   0   0   0   0  21   0   0   0   0   0   0   0   0   3   0   0   4]
 [  4   0   0   1   0   1  48   0   0   0   2   0   0   0   6   0   1   3]
 [  0   0   0   0   0   0   0   5   0   0   2   0   0   0   0   0   0   2]
 [  0   0   0   2   0   0   1   0  42   0   1   0   0   0   0   0   1   5]
 [  0   0   0   0   0   0   1   0   0   4   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   2   0   0   0 120   0   0   0   1   1   0   4]
```

```
 [  1   0   0   0   0   0   0   0   0   0   2  13   0   0   0   0   0   0]
 [  0   0   0   3   0   0   0   0   0   0   1   0   2   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   1   0   0   6   0   0   0   0]
 [  3   0   0   1   0   1   2   0   1   0   0   0   0   0  56   0   0   6]
 [  1   0   0   1   0   0   1   0   1   0   0   0   0   0   0  18   0   1]
 [  1   0   0   0   0   1   0   0   0   0   0   0   0   0   1   0  11   7]
 [  3   0   0   0   0   2   3   0   1   0   0   0   0   0   1   0   0
146]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.92      0.90       105
           1       1.00      0.80      0.89        10
           2       1.00      0.50      0.67         4
           3       0.83      0.91      0.86        53
           4       0.00      0.00      0.00         1
           5       0.78      0.72      0.75        29
           6       0.79      0.73      0.76        66
           7       0.83      0.56      0.67         9
           8       0.93      0.81      0.87        52
           9       1.00      0.57      0.73         7
          10       0.90      0.94      0.92       128
          11       1.00      0.81      0.90        16
          12       0.67      0.33      0.44         6
          13       1.00      0.86      0.92         7
          14       0.79      0.80      0.79        70
          15       0.95      0.78      0.86        23
          16       0.85      0.52      0.65        21
          17       0.80      0.94      0.86       156

    accuracy                           0.85       763
   macro avg       0.83      0.69      0.75       763
weighted avg       0.85      0.85      0.84       763


ROC AUC Score (One-vs-Rest): 0.9610364123132078


/opt/anaconda3/envs/pytorch_env/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/pytorch_env/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/pytorch_env/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Logistic Regression

```
import json
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# --- Load Node Features ---
# The JSON file contains a dictionary mapping node IDs (as strings) to
lists of artist IDs.
features_path =
'/Users/log/Github/Spring2025Classes/social_networks/project4/lasftm_asia/
lastfm_asia_features.json'
with open(features_path, 'r') as f:
    features_dict = json.load(f)

# Convert keys to integers and sort by node id for consistency.
features_dict = {int(k): v for k, v in features_dict.items()}
features_series = pd.Series(features_dict).sort_index()

# --- Convert Features to a Binary Matrix ---
# Here we treat each node's list of liked artists as a set of labels
# and convert it into a multi-hot (binary) feature vector.
mlb = MultiLabelBinarizer(sparse_output=True)
X = mlb.fit_transform(features_series)
print(f"Converted features to a binary matrix of shape: {X.shape}")

# --- Load Node Targets ---
# The CSV file contains two columns: 'id' and 'target'.
targets_path =
'/Users/log/Github/Spring2025Classes/social_networks/project4/lasftm_asia/
lastfm_asia_target.csv'
targets_df = pd.read_csv(targets_path)
targets_df.set_index('id', inplace=True)

# Align targets with the order of node IDs in our features.
targets_df = targets_df.loc[features_series.index]
y = targets_df['target']

# --- Split the Data ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# --- Train a Classifier ---
# We use Logistic Regression which can work directly with sparse input.
clf = LogisticRegression(max_iter=1000, solver='liblinear')
clf.fit(X_train, y_train)

# --- Evaluate the Model ---
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# --- Compute the ROC AUC Score (One-vs-Rest) ---
# Binarize the true labels
classes = sorted(y.unique())
y_test_binarized = label_binarize(y_test, classes=classes)

# Get the predicted probabilities for each class
y_score = clf.predict_proba(X_test)

# Compute the ROC AUC score using One-vs-Rest strategy
roc_auc = roc_auc_score(y_test_binarized, y_score, multi_class="ovr",
average="macro")
print("One-vs-Rest ROC AUC Score:", roc_auc)
```

```
Converted features to a binary matrix of shape: (7624, 7842)
Accuracy: 0.7632786885245901

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.86      0.86       225
           1       0.00      0.00      0.00         6
           2       0.75      0.27      0.40        11
           3       0.71      0.75      0.73        96
           4       0.00      0.00      0.00         2
           5       0.75      0.74      0.75        78
           6       0.67      0.68      0.67       142
           7       0.80      0.17      0.29        23
           8       0.87      0.70      0.77        93
           9       1.00      0.11      0.20         9
          10       0.81      0.88      0.84       269
          11       0.80      0.70      0.74        23
          12       0.33      0.11      0.17         9
          13       1.00      0.44      0.61        16
          14       0.72      0.65      0.68       117
          15       0.85      0.63      0.72        46
          16       0.83      0.58      0.68        50
          17       0.71      0.90      0.79       310

    accuracy                           0.76      1525
```

```
   macro avg       0.69      0.51      0.55      1525
weighted avg       0.77      0.76      0.75      1525

One-vs-Rest ROC AUC Score: 0.9241175188198765


/opt/anaconda3/envs/pytorch_env/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/pytorch_env/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/envs/pytorch_env/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```