MBTI Personality Types Classification Using GAT

Aruzhan Shinbayeva, Mariia Shmakova, and Alsu Khairullina

November 29, 2024

I Introduction

A. *Project Idea*

The Myers-Briggs Type Indicator (MBTI) is a popular personality assessment tool that categorizes individuals into 16 distinct personality types based on four dichotomies: Extraversion (E) vs. Introversion (I), Sensing (S) vs. Intuition (N), Thinking (T) vs. Feeling (F), and Judging (J) vs. Perceiving (P).

The goal of this project is to classify individuals into their respective MBTI personality types using Graph Machine Learning techniques, leveraging social network data and textual content.

B. *Dataset*

Twitter MBTI Personality Types Dataset

The dataset contains information about 8,328 users, their MBTI personality types in the profile description, and their social media posts. It includes features such as user ID, personality type (e.g., ENTP, ENFP), the content of the publication (text) and their subscriptions.

We chose this dataset because it contains a wealth of textual content and social interactions, making it a perfect candidate for applying graph machine learning techniques. This dataset allows us to examine the relationship between social network interactions and personality traits, providing valuable insights into personality analysis and social network research.

C. *Prediction Tasks*

Classify users into their respective MBTI personality types based on their interactions on social networks and textual content.

D. *Application Domain*

1. **Social Media Personalization**: Platforms could use personality predictions to suggest content or friends that match their interests or personality.

2. **Team Building**: Employers could use this to match people with jobs or teams where they'd thrive, based on their personality traits.

3. **Self-awareness and Personal Growth**: People can gain a better understanding of themselves by receiving predictions about their MBTI personality type based on their social media behavior. This can help guide personal development and self-improvement.

II Overview of the method

We use Graph Attention Network (GAT) for node classification. We will create a graph in which nodes represent users and edges represent social interactions such as subscriptions.

The GAT model is designed to learn node embeddings by aggregating information from neighboring nodes using attention mechanisms. The attention mechanism allows the model to weigh the importance of different neighbors, making it suitable for capturing complex relationships in social networks.

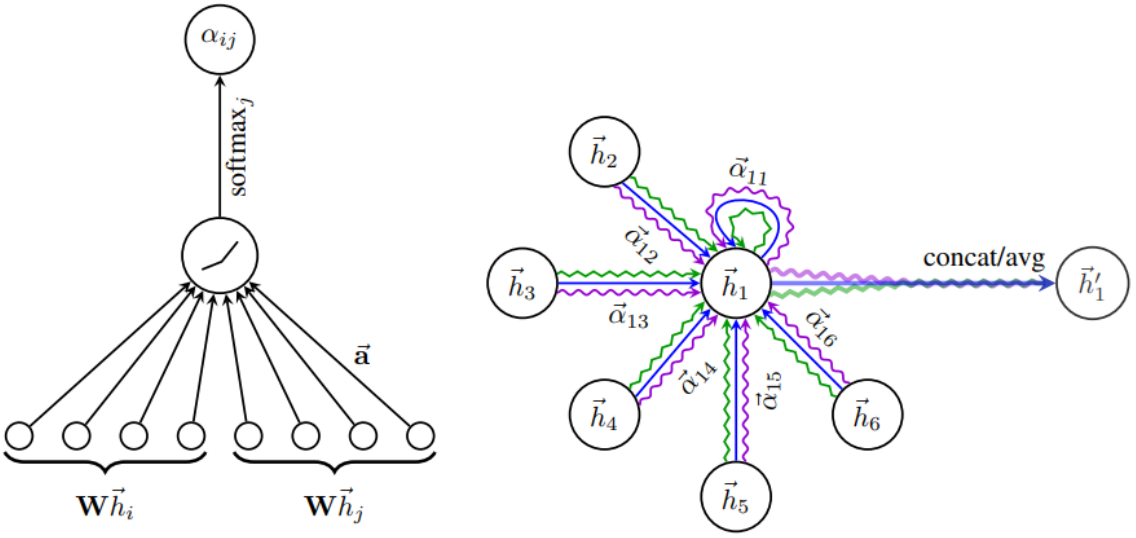


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Equations

1. Attention Mechanism:

$$e_{ij} = \text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j])$$

where $\mathbf{a}$ is the attention vector, $\mathbf{W}$ is the weight matrix, $\mathbf{h}_i$ and $\mathbf{h}_j$ are the embeddings of nodes i and j, and $\parallel$ denotes concatenation.

2. Attention Coefficients:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

where $\mathcal{N}_i$ is the set of neighbors of node i.

3. Node Embedding Update:

$$\mathbf{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right)$$

where σ is the activation function (e.g., ReLU).

III Description of the Dataset

The dataset used in this project consists of several CSV files, each containing different types of information related to Twitter users and their tweets. The primary goal is to predict users' Myers-Briggs Type Indicator (MBTI) personality types based on their Twitter activity. Below is a detailed description of each dataset:

A. `user_info.csv`

Description: This file contains additional information about each user, such as their profile description, follower count, and other metadata.

Columns: The columns contain information about the personal ID, name, nickname, location and some behavioral features in the tweeter, such as the number of subscribers, the number of posts, the number of likes, etc.

Example:

	id	id_str	name	screen_name	location	description	verified	followers_count	friends_count	listed_count	...	total_mentions_count
0	160881623	160881623	Biam 🌿 32 Days AC 🌴 🌷	_AiBiam	Hateno Village	🐸 {INFP} {ESP/ENG} • Current obsession: Unchart...	False	1904	782	67	...	139

B. user_tweets.csv

Description: This file contains the tweets of various Twitter users. Each row corresponds to a user, and each column represents a tweets from that user.

Columns: The columns contain information about the personal ID and the the text of individual tweets.

Example:

	id	tweet_1	tweet_2	tweet_3	tweet_4	tweet_5	tweet_6	tweet_7	tweet_8	tweet_9	...	tweet_191
0	160881623	@andresitonieve Me he quedado igual estoy llor...	RT @heikala_art: Fragment of a Star 💫 Celebrat...	RT @bananamisart: I heard it was BOtW's 3rd an...	RT @night_sprout: new banner time!! https://t....	RT @dealer_rug: Why is everyone buying toilet ...	@andresitonieve Amo el diseño de este personaje	RT @Tchaigothsky: UNFORTUNATELY I CANT STOP WA...	RT @_Ritao_: IT'S SO CUTE AHHHHHHH 😭 https://t....	RT @Lesfleursdmal: Os dejo esto por aquí por s...	...	NOOOOOOOOOOO MURCIA HA CAÍDO TAMBIÉN 😔😔😔

C. mbti_labels.csv

Description: This file contains the MBTI personality type labels for each user.

Columns: The columns contain information about the personal ID and the MBTI personality type label for the user (e.g., INFP, ENTP).

Example:

	id	mbti_personality
0	160881623	infp

D. *edges.csv*

Description: This file contains ordered pairs of user IDs depicting a follower-followee relationship on Twitter.

Columns: The columns contain information about the personal IDs of followers and followees.

Example:

	follower_id	followee_id
0	5660312	1654576440

IV The roadmap of the model creation

A. *Loading data and deleting null values*

- The datasets are loaded into Pandas DataFrames.
- Missing values are removed or filled as necessary.
- The `id` column is converted to string type for consistency.

Figure 1: The process of loading and cleaning data.

```
tweets_df = pd.read_csv('user_tweets.csv', dtype='unicode')
labels_df = pd.read_csv('mbti_labels.csv')
info_df = pd.read_csv('user_info.csv')
edges_df = pd.read_csv('edges.csv')

# Remove missing values
tweets_df = tweets_df.dropna()
info_df = info_df.dropna()

# Convert 'id' column to string type
tweets_df['id'] = tweets_df['id'].astype(str)
labels_df['id'] = labels_df['id'].astype(str)
info_df['id'] = info_df['id'].astype(str)
```

B. *Data Cleaning and Preprocessing*

Initially, we clean up our data by removing URLs, mentions, numeric data, hashtags, punctuation marks and stop words.

Figure 2: Text Cleaning

```
tweets_df = pd.read_csv('user_tweets.csv', dtype='unicode')
labels_df = pd.read_csv('mbti_labels.csv')
info_df = pd.read_csv('user_info.csv')
edges_df = pd.read_csv('edges.csv')

# Remove missing values
tweets_df = tweets_df.dropna()
info_df = info_df.dropna()

# Convert 'id' column to string type
tweets_df['id'] = tweets_df['id'].astype(str)
labels_df['id'] = labels_df['id'].astype(str)
info_df['id'] = info_df['id'].astype(str)
```

Then, we apply feature engineering techniques:

- TF-IDF vectorization was then applied to the text data from the dataset to extract meaningful elements from the tweets.

Figure 3: Creation of text embeddings

```
tweets_df["all_tweets"] = tweets_df[tweet_columns].apply(lambda x: " ".join(x.fillna("").astype(str)), axis=1)

charngram_tfv = TfidfVectorizer(strip_accents='unicode', analyzer='char', sublinear_tf=1)
charngrams = charngram_tfv.fit_transform(tweets_df["all_tweets"])

charngrams_df = pd.DataFrame.sparse.from_spmatrix(charngrams)
charngrams_df['id'] = tweets_df['id']
```

- Min Max Scale was applied to the numerical data from the dataset to shrink the data within the given range.

Figure 4: Normalization

```
# Normalize numerical features
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical_cols = ['followers_count', 'friends_count', 'listed_count'

data[numerical_cols] = scaler.fit_transform(data[numerical_cols])
data.head()
```

- LabelEncoder was applied to the labels from the dataset to convert categorical columns into numerical ones.

Figure 5: Labels encoding

```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()
labels_df['mbti_personality'] = label_encoder.fit_transform(labels_df['mbti_personality'])
```

We also merge our dataframes into a single one.

Figure 6: Data Merging

```
data = pd.merge(data, charngrams_df, on="id")
```

C. *Graph Construction*

A graph is constructed where the nodes represent the users and the edges represent the social connections. This graph is then used as input for a graph neural network (GNN).

Figure 7: Graph Construction

```
import networkx as nx
import pandas as pd

G = nx.DiGraph()
G.add_edges_from(edges_df.values)

print(f"Number of nodes in the graph: {G.number_of_nodes()}")
print(f"Number of edges in the graph: {G.number_of_edges()}")

labeled_nodes = set(data['id'])
graph_nodes = set(G.nodes)

# Find the nodes that are in the data, but are missing in the graph
missing_nodes = labeled_nodes - graph_nodes
print(f"Missing nodes (in data but not in graph): {len(missing_nodes)}")


for node in missing_nodes:
    G.add_node(node)

print(f"Total number of nodes after adding missing ones: {G.number_of_nodes()}")

isolated_nodes = list(nx.isolates(G))
print(f"Number of isolated nodes: {len(isolated_nodes)}")
```

```
Number of nodes in the graph: 6067
Number of edges in the graph: 36466
Missing nodes (in data but not in graph): 826
Total number of nodes after adding missing ones: 6893
Number of isolated nodes: 826
```

D. *Model Training*

The model training phase in our project involves several key steps:

1. **Graph Conversion**: Converting the NetworkX graph into a format suitable for PyTorch Geometric, a library for deep learning on graphs.
2. **Data Splitting**: Splitting the data into training and testing sets.
3. **Model Definition**: Defining the graph neural network (GNN) model using GATConv layers.
4. **Training the Model**: Training the GNN model on the training data.
5. **Evaluation**: Evaluating the model's performance on the test data.

Graph Conversion: First, we need to convert the NetworkX graph into a PyTorch Geometric Data object, which is the standard format for GNN inputs.

Figure 8: Graph Conversion

```
pyg_graph = from_networkx(G)
pyg_graph.x = torch.tensor([G.nodes[node]['features'] for node in G.nodes], dtype=torch.float)
pyg_graph.y = torch.tensor([G.nodes[node]['label'] for node in G.nodes], dtype=torch.long)
```

Data Splitting: We split the data into training and testing sets using a train-test split. We also create masks to indicate which nodes belong to the training set and which belong to the test set.

Figure 9: Splitting into train and test

```
# Train-test split
train_ids, test_ids = train_test_split(data['id'], test_size=0.2, random_state=42)
pyg_graph.train_mask = torch.tensor([node in train_ids for node in G.nodes], dtype=torch.bool)
pyg_graph.test_mask = torch.tensor([node in test_ids for node in G.nodes], dtype=torch.bool)
```

Model Definition: We define a Graph Neural Network (GNN) model using PyTorch Geometric. The model consists of several GATConv layers followed by a fully connected layer for classification.

Figure 10: GNN Stack with GATconv layer

```python
import torch
import torch.nn as nn
from torch_geometric.nn import global_mean_pool
from torch_geometric.utils import to_dense_batch
from torch_geometric.nn import GATConv


class GNNStack(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, args, emb=False):
        super(GNNStack, self).__init__()

        self.args = args

        self.gat0 = GATConv(input_dim, hidden_dim, heads=args['heads'], dropout=args['dropout'])
        self.gat1 = GATConv(hidden_dim * args['heads'], hidden_dim, heads=args['heads'], dropout=args['dropout'])
        self.gat2 = GATConv(hidden_dim * args['heads'], output_dim, heads=args['heads'], dropout=args['dropout'])

        self.fc = nn.Linear(hidden_dim * args['heads'], output_dim)

    def forward(self, data):

        x, edge_index = data.x, data.edge_index
        x = F.relu(self.gat0(x, edge_index))
        x = F.relu(self.gat1(x, edge_index))
        x = self.gat2(x, edge_index)

        return F.log_softmax(x, dim=1)

    def loss(self, pred, label):
        return nn.functional.nll_loss(pred, label)
```

Training the Model: We initialize the model, define the loss function (negative log likelihood loss), and the optimizer (AdamW). Then, we train the model for a specified number of epochs.

Figure 11: Training the model

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

input_dim = pyg_graph.x.shape[1]
output_dim = len(label_encoder.classes_)
model = GNNStack(input_dim=input_dim, hidden_dim=64, output_dim=output_dim,
                 args={'heads': 4, 'dropout': 0.6}).to(device)


# Train GNN model
optimizer = AdamW(model.parameters(), lr=0.005)
pyg_graph = pyg_graph.to(device)


num_epochs = 1000
best_accuracy = 0.0

for epoch in range(num_epochs):

    loss = training_step(model, optimizer, pyg_graph)
    accuracy = test(model, pyg_graph)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
    if epoch%100==0:
        print(f"Epoch {epoch}/{num_epochs} - Loss: {loss:.4f}, Test Accuracy: {accuracy*100:.2f}%")

print(f"Best Test Accuracy: {best_accuracy*100:.2f}%")
```

V Results

A. *Dataset and Preprocessing*

We used an extensive dataset that includes tweets from international users and their user information.

Several data stripping experiments were conducted to improve the metric. Several techniques were tested, including stripping all stop words, links, hashtags, numerical data, emoji and other things in the data. It was found to be the most effective to leave emoji, since they most characterize the type of personality, and remove all the rest of the above.

Another feature of the dataset was also revealed - it contains more than 6 languages. An experiment was also conducted during which it was found out that it does not make much difference for the model to work with translated text into one language, or use several.

B. Model Architecture

Our model is based on a Graph Attention Network (GAT), which leverages attention mechanisms to weigh the importance of neighboring nodes in the graph. The GAT model was trained on the constructed graph to predict the MBTI personality type of each user. The model was trained using a loss function (negative log likelihood loss), and the optimizer (AdamW).

C. Evaluation Metrics

We used the following evaluation metrics to assess the performance of our model:

- **Accuracy:** The proportion of correctly predicted personality types.
- **Loss:** The negative log likelihood loss, which measures the difference between the predicted and actual labels.

D. Achieved results

The proposed GAT model has demonstrated a significant increase in accuracy compared to the training stages. When translating into English or deleting individual nodes, the results slightly decreased to 86% and 85%, respectively.

The model achieved the best test accuracy - **88. 19%** without removing isolated nodes and translating the text into English. The results of our experiments are shown below.

Figure 12: Training the model with deletion of isolated nodes

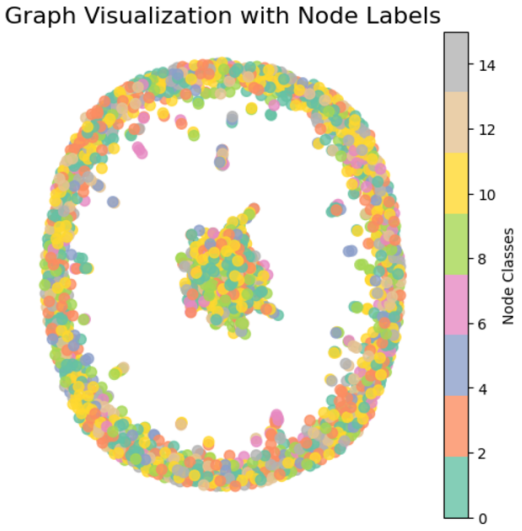
```
Epoch 0/1000 - Loss: 4.1518, Test Accuracy: 10.87%
Epoch 100/1000 - Loss: 2.2796, Test Accuracy: 60.14%
Epoch 200/1000 - Loss: 2.0078, Test Accuracy: 74.65%
Epoch 300/1000 - Loss: 1.8434, Test Accuracy: 78.02%
Epoch 400/1000 - Loss: 1.7969, Test Accuracy: 80.92%
Epoch 500/1000 - Loss: 1.7390, Test Accuracy: 81.37%
Epoch 600/1000 - Loss: 1.6806, Test Accuracy: 84.53%
Epoch 700/1000 - Loss: 1.7513, Test Accuracy: 85.07%
Epoch 800/1000 - Loss: 1.6622, Test Accuracy: 86.37%
Epoch 900/1000 - Loss: 1.6413, Test Accuracy: 87.00%
Best Test Accuracy: 88.19%
```

E. Visualization

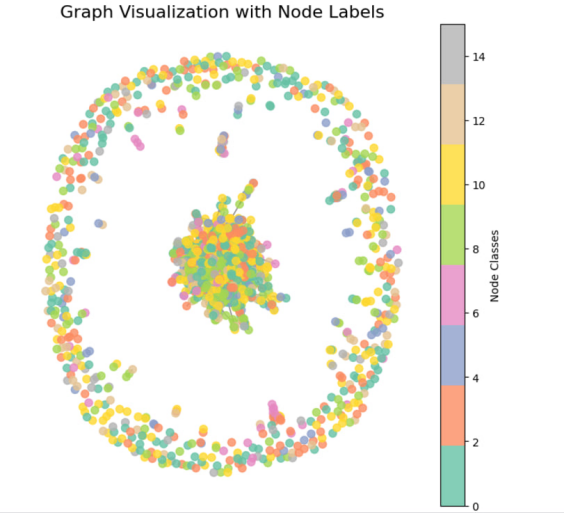
We plot the graph using node features and labels. We use the Spring Layout algorithm to place the nodes, which mimics the forces that would distribute them evenly. We also added a color bar to visually identify the different node classes.

During the experiments with data preprocessing, the following graphs were constructed:

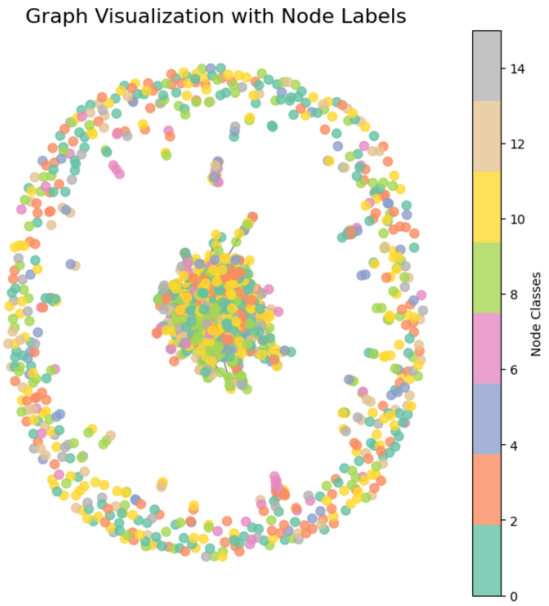
- A graph **without** deleting isolated nodes and **without** translating the entire text into English.



- A graph **with** deleting isolated nodes and **without** translating the entire text into English.



- A graph **with** deleting isolated nodes and **with** translating the entire text into English.



As a result, after analyzing the graphs, it was revealed that the translation of the entire text into one language does not play a big role both in the structure of the graph and in the resulting metric based on the results of the trainings. Isolated nodes are mostly located in the outer ring of the graph and during training, the removal of isolated nodes played a minor role (it fell exactly by 0.02).

Examples of work

As an example of our work, we collected information from the VK social network of one of our team members.

Figure 13: User's data

```
# Provided data
user_data = {
    "name": "Mariia Shmakova",
    "screen_name": "marishmak",
    "location": "Innopolis\\Russia",
    "description": "Data science\\Innopolis university",
    "followers_count": 58,
    "friends_count": 248,
    "listed_count": 21,
    "favourites_count": 2000,
    "statuses_count": 6,
    "number_of_quoted_statuses": 1,
    "number_of_retweeted_statuses": 2,
    "total_retweet_count": 2,
    "total_favorite_count": 246,
    "total_hashtag_count": 11,
    "total_url_count": 3,
    "total_mentions_count": 0,
    "total_media_count": 6,
    "average_tweet_length": 233,
    "average_retweet_count": 0.3,
    "average_favorite_count": 41,
    "average_hashtag_count": 1.8,
    "average_url_count": 0.5,
    "average_mentions_count": 0,
    "average_media_count": 1,
    "tweets": [
        "photo_update",
        "The final of the "Flying Robotics" profile of the student
        "photo update",
        "photo update",
        "photo update",
    ]
}
```

Preprocessing the data, adding them to the node, and getting a prediction:

```
print(f"Predicted MBTI Personality: {predicted_mbti}")
```

```
Predicted MBTI Personality: intj
```

VI Conclusion

Our experiments demonstrate that the GAT model achieves an outstanding test accuracy of 88.19%, outperforming other models in predicting MBTI personality types based on social network data. The ability of this model to utilize attention mechanisms in order to understand intricate relationships within a graph significantly contributes to its impressive performance.

In summary, our approach represents a significant advancement in the field of personality prediction using social media data, establishing a new benchmark for this task.