Point of Interest Recommendation using Graph Neural Networks

Rossella Tritto

Tools

- ResNet and BERT Model
- Graph Neural Network: GATConv
- PyTorch and PyTorch Geometric
- Scikit-learn, Pandas, Numpy

1. Introduction

This report presents the developement of a Graph Neural Network designed to recommend Points of Interest (POIs) to users. In this context, POIs refer to specific locations or attractions such as restaurants, historic sites, or entertainment venues. By using information about the POIs and check-in data of users, the GNN is trained to understand the relationships between users and POIs and generates personalized recommendations for each user. This approach uses a graph representation, where nodes correspond to users and POIs and edges represent the interactions (check-ins) between them. Through this structure, the GNN learns from the data and suggests relevant POIs.
The following sections detail the methods used, the results obtained and the challenges encountered in the implementation, together with suggestions for future improvements.

2. Methodology

This section discusses the choices made for the implementation, focusing on dataset preprocessing, obtaining information about POIs, embeddings extraction, model architecture, training and evaluation.

2.1 Dataset Preprocessing

Regarding the dataset, it was obtained from Foursquare. The dataset includes check-in data collected from Foursquare from April 2012 to September 2013. It contains three files: one with check-ins information including 'UserID', 'VenueID', 'UTC time' and 'Timezone offset', another file with city information and one with venue details including 'VenueID', 'Latitude', 'Longitude', 'Venue category name' and 'Country code'.
First, the dataset was reduced by narrowing the analysis to a single country, resulting in all POIs with CountryCode='FR'. Since the dataset was still quite large (approximately 10,000 POIs), the analysis was further restricted to the 1st arrondissement of Paris, selecting POIs that fell within an approximate range of latitude and longitude. This restriction was also applied to the check-ins,

which were filtered to include only those related to the selected POIs. Basically, 1,491 POIs and 9,485 check-ins have been used.
This preprocessing was performed using the script `dataset_preprocessing.py`. Additionally, the Longitude and Latitude columns were scaled between 0 and 1 using MinMaxScaler to normalize the values for subsequent operations.

Next, in the file `request.py`, information about the POIs were extracted from Foursquare. Specifically, a request was made to the Foursquare API, which returned a JSON with various details about the specified venue. I chose to extract 'Name', 'Description' and 'BestPhoto' and store the results in a dictionary for each VenueID. Due to rate limits imposed by Foursquare, I implemented `get_descriptions` to pause for 1 second every 50 requests (*Rate Limit of 50 QPS*) and `save_descriptions` to batch 500 requests per hour (*Rate Limit Per Endpoint Resource: 500 per hour per OAuth token*).
During the process, the following errors were encountered:

```
Error 400: Value 4c7c9957a856dcbb1fbe2a7 is invalid for venue id
Error 400: Value 404c64065bf69431df9620 is invalid for venue id
```

As a result, these two VenueIDs were excluded. The obtained information was saved in a file named `pois_information.csv`, which was merged with the previous POI file to obtain complete information. For simplicity, the `pois_information.csv` file is already available to avoid waiting approximately 2 hours to retrieve the information. This file includes 'VenueID', 'Latitude', 'Longitude', 'Venue_category_name', 'Name', 'Description' and 'BestPhoto'.
The description field contains details about services offered, location, significance and other specifics based on the category. Some examples are shown below:

Table 1. POIs Information

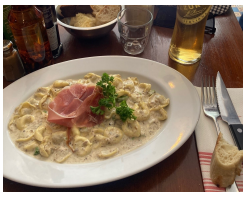
Name	Venue Category Name	Description
Eiffel Tower	Historic Site	Dominant Paris du haut de ses 324 mètres, la Tour Eiffel fut construite de 1887 à 1889 pour l'exposition universelle de 1889. Aujourd'hui, elle est le symbole de la France et le monument le plus visité au monde
Hôtel Britannique	Hotel	Chambres élégantes et feutrées avec wifi et double vitrage dans un bâtiment haussmannien à la déco romantique
Gambino	Italian Restaurant	Carte étoffée dans une salle de style maison de campagne italienne avec ses pierres et poutres apparentes
Chaumet	Jewelry Store	The Chaumet store offers you jewels crafted from gold, platinum and diamonds... Rings, wedding bands, necklaces, pendants, bracelets, earrings in Paris
Louvre Museum	History Museum	Plus grand musée de France, le Louvre est un ancien palais royal qui accueille aujourd'hui une riche collection d'art du monde entier

The images can vary: restaurants may have pictures of courses, hotels may show rooms or

hotel locations and products sold inside stores. For example the extracted images of the previous venues are in Figure 1


Eiffel Tower


Hôtel Britannique


Gambino


Chaumet

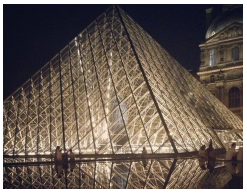

Louvre Museum

Figure 1. POIs Images

2.2 Embeddings Extraction

The `pois_information.csv` dataset was used to obtain the embeddings of the textual information and images through `venue_embeddings.py`.

2.2.1 Image Embeddings

For the images, the **ResNet-50** model from the Torchvision library was used to obtain the embeddings. There are several variants which increase in complexity. However, I decided to use the 50-layer version because it is still a complex model that captures features well but requires fewer computational resources compared to model with more parameters.

For the model, I used the default pre-trained weights, that through the `transform()` function, enable preprocessing operations on the images. With the default weights, the operations include: resizing the image, central cropping, rescaling values and normalization with specified mean and standard deviation.

In `get_image_embeddings`, a try-except block was added because during execution, I encountered frequent errors related to ReadTimeout, even though the timeout parameter in the GET request had already been increased to 10 seconds. For this reason, if the request to fetch the image from the URL takes too long (`requests.exceptions.Timeout`) or if there are other errors related to the request, it waits a few seconds and then tries again. If the maximum number of allowed attempts is reached, it raises the exception. With this strategy, these types of errors no longer occurred, so my problem was probably a weak connection that needed more time to process the request. However, after preprocessing the images, they are passed to the model to obtain the feature embeddings.

2.2.2 Text Embeddings

For textual information, **BERT** was used. In this case as well, before passing the inputs to the model, they are processed through a pre-trained tokenizer that transforms the inputs into a format suitable for the model. Specifically, the Name, Venue Category Name and Description fields were first aggregated using the [SEP] token to separate the fields:

```
'{desc} [SEP] {name} [SEP] {category}'
```

Next, the tokenizer splits the text into tokens, ensures that all sequences have the same length by adding zeros (padding), truncates sequences that are too long and finally converts the tokens into

IDs, giving in output the necessary parameters for the model. BERT transforms the inputs into embeddings, from which only the embedding corresponding to the [CLS] token is extracted as representation of the entire text for each input row.

Finally, the image and text embeddings are concatenated together so they can be used as features in the model and during training they will be included as parameters for fine-tuning.

2.3 Model Architecture

The goal of the project is to implement a GNN for POI recommendation that captures interactions between users and venues and predicts which venues a user is likely to visit. I used the PyTorch Geometric library, which offers various types of Convolutional Layers. To obtain the outputs from this library, using the forward method, it is necessary to transform the available data into a suitable format, specifically a Data Object. It describes a homogeneous graph and it is formed by a node feature matrix, an edge index and a tensor for the ground truth. Then, before implement the model, these data structures are obtained.

2.3.1 Node Feature Matrix

Since the goal is to recommend POIs to users, the nodes will represent both users and venues: specifically, the first nodes represent users and the subsequent ones represent venues. Then, the node feature matrix has dimensions `[num_nodes, num_node_features]`. I obtain POI embeddings that were computed before and, to further enrich them, I added geographical context by concatenating 'Longitude' and 'Latitude' values. Since users are also nodes, I initialized user features randomly, creating them with the same dimensions as the POI features.
All features are concatenated to form the feature matrix, which is then converted into a tensor to fit the format required. Specifically, the tensor is created with `requires_grad=True` to allow the model to learn these features during training.

2.3.2 Edge Index

The edge index tensor represents the connections between nodes in the graph, where each connection is an interaction between a user and a venue. This tensor has a shape of `[2, num_edges]`, with the two rows containing the source and target node indices. The check-in data is used to populate the edge index: each row represents an edge (link) from a user to a venue.

2.3.3 Negative Edge Generation

Since the edge index is only composed of positive samples (check-in data), to balance the dataset, negative edges are generated. These represent user-POI pairs that do not exist in the original check-in data. To do this, I randomly select a user and a POI that exists in the dataset, and if this pair does not exist in the set of edges, I add it to the list of negative edges. The resulting negative edges are then transformed to match the structure of the edge index tensor.
The choice of the size negative samples is critical: to balance the dataset I decided to generate negative samples in the same number of positive samples.

2.3.4 Ground Truth Labels

For the Data Object, labels are also needed. I assigned labels to the edges: 1 for existing edges (from the original edge index in 2.3.2) and 0 for negative edges (computed in 2.3.3). These three

components, node features, edge index (with negative edges) and labels, are combined into a Data object.

2.3.5 Splitting

To train and evaluate the model, different datasets must be created. The data is split into 70% for training, 10% for validation and the remaining 20% for testing. The splitting is done by randomly selecting indices from the edge index tensor and a data object is created for each split.

2.3.6 GNN Model

The main component is the GNN architecture, specifically I used the **Graph Attention Network** (GAT) convolutional layer. I chose this layer due to its performance: during experimentation with different architectures and parameters, I found that GAT provided higher accuracy compared to the standard GCN. This is most likely due to the fact that the attention mechanism of GAT allows to better capture the important relationships between users and POIs, in fact it dynamically weights the contributions of neighboring nodes when aggregating the information. On the contrary, GCN assigns the same weight to all neighbors, which may lead to a lower ability to capture the most relevant information.

The overall architecture consists of a GAT convolutional layer, a dropout layer that randomly set some elements to zero during training, a fully connected layer for classification and a softmax function to obtain probabilities. Although I tried with several options, adding more convolutional layers, the best option was the one with only one GAT layer.
The previously obtained embeddings are included in the model's parameters so that they are learnable during training and they will be updated during backpropagation. The GAT layer processes these embeddings to capture relationships between nodes by multiplying the features of each pair of nodes.
The output of the GAT layer is then passed through a dropout layer to reduce overfitting.
Since the objective is to perform link prediction (determine if an edge exists between two nodes), a fully connected layer is applied, followed by a log softmax function to output the probabilities of a link between user and POI.

2.4 Training and Evaluation

The model was trained for 30 epochs using the training set, with the **AdamW** optimizer and the **NLLLoss** function, given that the model outputs are in log softmax. The validation set was used to find the best parameter configuration. Specifically, a Grid Search was employed to identify the optimal values for three parameters: the learning rate and weight decay of the optimizer and the number of hidden channels for the model. Instead of using accuracy, I chose the Average Precision (AP) as criterion for determining the best configuration. AP summarizes the precision-recall curve as the weighted mean of precisions achieved at each threshold and it is preferred in a recommendation system where the ranking of positive items is important.
Once the best parameters were identified, the model was evaluated on the test set using several important metrics: accuracy, precision, recall, F1 score, average precision and NDCG score. Additionally, the ROC curve and AUC score were computed and the results were plotted. Using the ROC curve, I also obtained the threshold that can be used in predictions: the optimal threshold is found by maximizing the difference between true positive rate and false positive rate. Then, this threshold is used to generate binary predictions for classification and these are given as input to

metrics like accuracy, precision, recall and F1 score which are based on class predictions.
In contrast, AP and NDCG (Normalized Discounted Cumulative Gain) are more relevant for recommendation systems as they use the model's probability scores as inputs. NDCG represents how well the system orders items according to their relevance, by considering their positions in the ranking, emphasizing the importance of placing more relevant POIs higher in the list. Even if in this case all check-ins are treated as equally valuable (in labels all checkins are considered equal, there are no more important ones that have a higher relevance), NDCG still provides a measure of how well the model ranks POIs.

2.4.1 User Recommendation

Since the metrics computed above do not provide deep information about the quality of the recommendations, to improve the evaluation I implemented `user_recommendations.py`, designed to generate and evaluate recommendations for each user.
The `get_recommendations` function constructs a list of recommended POIs for each user based on the model's output probabilities and stores the recommendations in a dictionary where each key corresponds to a user and the value is a sorted list of POIs with their probabilities.
The `get_recommendation_labels` function stores the ground truth POIs, identifying the actual POIs each user checked in based on the value of the labels. There is also a function `print_recommendations` to print the top recommended POIs and their true checkin for each user.
Finally, `compute_metrics` evaluates the quality of the recommendations by computing key metrics for a recommendation system:

- **Precision@K**: measure the accuracy of the top K recommendations. It is computed dividing the number of true positives POIs (recommended POIs that are in the ground truth) among the top K by K. This metric reflects how many of the top K recommendations are actually relevant for the user. As will be seen in the results, the choice of parameter k highly influences this metric;
- **Recall@K**: evaluate how well the system captures all relevant POIs within the top K recommendations. It is computed by dividing the number of true positives among the top K by the total number of relevant POIs for that user;
- **MRR** (Mean Reciprocal Rank): focus on the position of the first relevant POI in the recommendation list. The Reciprocal Rank is the inverse of the rank position of the first relevant POI.

All these metrics are computed for each user, then the results are summed up together and averaged to get the final metric value. It is important to note that I decided to compute these metrics only for users who had POIs in the ground truth, otherwise it would not have made sense to compare the results. Likewise, the average was not done on the total number of users but only on the number of users who had the ground truth

3. Results and Discussion

The model was trained and evaluated using a T4 GPU provided by Google Colab. Through the hyperparameter tuning the best hyperparameters obtained are: **Hidden Channels**: 32, **Learning**

Rate: 0.001 and **Weight Decay**: 0.01. These hyperparameters yielded the highest average precision of 0.82. Below there are the detailed results at various epochs:

```
Epoch 10, Loss:  0.6703, Val Loss:  0.6672, Average Precision:  0.7012
Epoch 20, Loss:  0.6603, Val Loss:  0.6435, Average Precision:  0.7589
Epoch 30, Loss:  0.6320, Val Loss:  0.6278, Average Precision:  0.8176
```

The best combination of hyperparameters effectively reduced the training and validation loss while improving the average precision, indicating that the model is actually performing well.

The resulting ROC Curve and the evaluation metrics on the testing data are:

Table 2. Metrics on testing data

Accuracy	Precision	Recall	F1 Score	Average Precision	NDCG Score
0.7356	0.7995	0.6376	0.7094	0.8376	0.9775

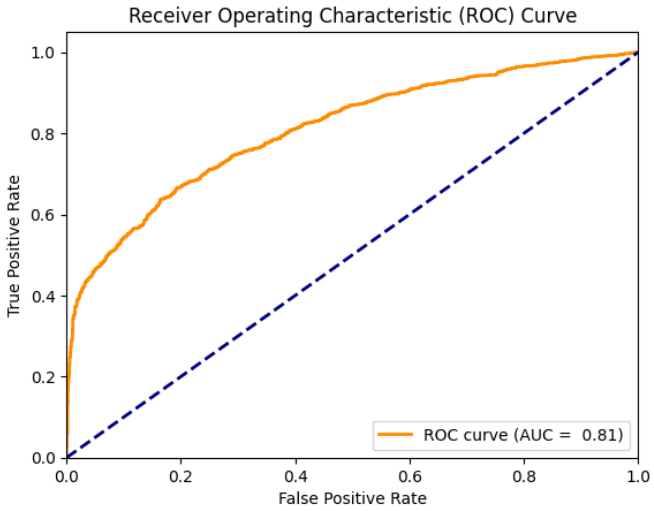


Figure 2. ROC Curve

These metrics indicate that the model performs well, with a high precision and a good balance between precision and recall, as seen from the F1 Score. The AUC score of 0.81 suggests that the model is good in distinguishing between positive and negative classes and the high value for the NDCG, near 1, means that the model is very effective in prioritizing the most relevant POIs.

Below, there are some examples of the recommendation list generated for each user, providing the top 3 recommended POIs along with their respective scores and their ground truth.

As can be seen, for users 4 and 9, the model perfectly predicts the recommended POIs with high confidence. For user 10, 2 predictions are correct, but there is an additional one. This, however, cannot be considered an error, as it is simply a way for the model to recommend POIs and the edge in question could be included in another set. On the contrary, a clear error is with user 8, as no POIs are predicted even though there is a check-in in the data. Therefore, while some

Table 3. Recommendation Lists for Users

User	Top 3 Recommended POIs	Ground Truth POIs
4	[(2658, 0.733), (2876, 0.515), (3456, 0.512)]	[2856, 3456, 2856, 3456, 2856, 2658, 2876]
5	[]	[]
7	[(3763, 0.519)]	[]
8	[]	[2950]
9	[(2548, 0.628), (3090, 0.539), (3348, 0.512)]	[3090, 3348, 2548]
10	[(2538, 0.776), (2508, 0.769), (2747, 0.505)]	[2508, 2538]

recommendations are highly accurate, others show limitations, especially when the ground truth list is empty.

The following metrics are computed for this recommendation system, summarizing the overall performance of the model in Table 4:

Table 4. Recommendation Accuracy Metrics

k=3		k=2		
Precision@3	**Recall@3**	**Precision@2**	**Recall@2**	**MRR**
0.3112	0.5552	0.4167	0.5306	0.6522

These metrics show that the model is moderately successful in recommending relevant POIs within the top 3 and top 2 positions. In particular, the model performs well in ensuring that relevant POI appear among the top recommendations given the relatively high MRR. However, as said before, the precision and recall are strongly influenced by the choice of k: while Precision@2 is higher than Precision@3, reflecting that as k increases, the proportion of relevant POIs in the top recommendations tends to decrease, the recall increases with the increasing of k, suggesting that more recommendations allow better capturing relevant items.

4. Conclusion

The results show that the model performs well, achieving high precision and recall, as well as strong NDCG and MRR scores, indicating its effectiveness in recommending relevant POIs. However, the development of the model presented several challenges, particularly in constructing the graph and creating negative edges, which may have impacted its overall performance. As mentioned earlier, I encountered difficulties in extracting information for the POIs due to rate limits imposed by the API. Another significant challenge was the data format, which made difficult implement the graph structure and create the correct labels for positive and negative samples. Additionally, during training, the model showed signs of overfitting, leading me to increase the dropout probability.

For future work, the model could be extended to other countries to test the generalization, although this was not possibile in my case due to the high computational costs. Additionally, specific features for users can be used, rather than initializing them randomly as it could improve the model's performance. Finally, another possible improvement could be a different edge splitting, that might provide a more robust evaluation of the recommendations, as the current approach could have influenced the results, particularly for users with few interactions.