Recommendation of Fitness Venues Using Graph Neural Networks

Pedro Marques
Instituto Superior Técnico
Lisbon, Portugal
pedro.marques.99@tecnico.ulisboa.pt

ABSTRACT

Gympass offers a different range of wellness products to its users: gyms, classes, personal trainers, and apps. But the main product is gyms. Users should be able to use the Gympass app to find recommendations for gyms, according to their personal preferences. Thus, we can pose the question: how to recommend gyms that are so distinct? Gympass is a subscription benefit that allows users to access multiple gyms in their area, but, if users only go to the same gym, they might unsubscribe Gympass. Therefore, we want to make sure the recommendation system (RS) is good at recommending new gyms so that they find the Gympass subscription useful. My M.Sc. project addresses the development and evaluation of graph neural network (GNN) approaches, specifically envisioning applications in the recommendation of Gympass gyms. Taking inspiration from previous work such as PinSage [14], GNN at Decathlon [4] and LARS [10], I implemented a similar approach and evaluated it on Gympass data. The results of our GNN RS seem promising for the case of recommending users to new gyms that they are visiting for the first time. The obtained results support the understanding that a deep learning model can recommend new Gympass gyms to users. The main contribution of this work relies on building and validating a RS based on GNN that infers how to model Gympass complex environment into a graph, using a GNN model architecture learns users' past behaviors and with the ranking function recommends gyms to users.

KEYWORDS

Recommendation System, Deep Learning, Graph Neural Networks, Machine Learning

ACM Reference Format:
Pedro Marques. 2022. Recommendation of Fitness Venues Using Graph Neural Networks. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Applications of RS increasingly rely on DL techniques to learn meaningful low-dimensional embeddings of images, text, and even users [3, 12]. DL representations can replace or enhance more conventional recommendation methods like collaborative filtering Systems (CF). Significant progress has been made in this area recently, particularly with the creation of new DL techniques that can learn from graph-structured data, which is essential for recommendation applications (e.g., to exploit user-to-item interaction graphs) [14].

Gympass offers a different range of wellness products to its users: gyms, classes, personal trainers, and apps. But the main product is gyms. Users should be able to use the Gympass app to find recommendations for gyms, according to their personal preferences. Different gyms may contain different information regarding their activities, description, location, and which is the minimum plan for a user to check-in there. Thus, we can pose the question: how to recommend gyms that are so distinct? Furthermore, Gympass is a subscription benefit that allows users to access multiple gyms in their area, but, if users only go to the same gym, they might unsubscribe Gympass and pay the subscription only to the gym they go to. In order to avoid this scenario, one of our concerns is to evaluate if the RS is able to recommend not only gyms that the users usually go to but also new gyms they haven't tried before. In other words, we want to make sure the RS is good at recommending new gyms so that the users try new gyms, besides the ones they already know, so that they find the Gympass subscription useful and keep paying for it.

The Gympass variety of data and the links between them makes a case for the use of graph techniques, more specifically GNN. Although they were first proposed in the late 1990s [11] and early 2000s [7], GNNs are now extensively used for a variety of tasks, including online and movie recommendations [17, 15]. Recent research shows that highly scalable GNNs for recommendation are possible [14]. The capacity of GNNs to represent non-Euclidean data is one of the factors driving such attention [16].

GNNs can be defined as neural networks that operate on graph data, in order to learn new embeddings for all graph features. These representations can have several practical applications. For instance, in the context of recommendation systems, these networks can be grouped into two scenarios depending on their application: (1) Non-structural scenarios where the relational structure is implicit or absent and generally include images and text; (2) Structural scenarios, where the data has an explicit relational structure. These second scenarios, on the one hand, often emerge from scientific research, such as graph mining, modeling physical systems, and chemical systems. On the other hand, they can also rise from applications such as knowledge graphs, traffic networks, and RS [18].

When considering efficient highly-scalable GNN algorithm to recommend items to users, it is necessary to consider more complex algorithms such as PinSage [14]. PinSage does not require operating on the full Laplacian graph during training because it

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

uses many techniques such as batching. The batching technique is used together with the re-indexing technique to create a sub-graph containing nodes and their neighborhood, which otherwise would not fit into memory. The task of generating embeddings outputs a representation of a node that incorporates both information about itself and its local graph neighborhood.

GNNs at Decathlon [4] builds upon the PinSage idea to recommend users items with GNN. The authors build a graph with nodes and edges, generate embeddings for each node and apply a max-margin loss function with a set of training edges and a set of nodes negatively sampled.

LARS [10] is a location-aware RS that uses location-based ratings to produce recommendations. LARS produce recommendations within reasonable travel distances by using travel penalty, a technique that penalizes the recommendation rank of items the further in travel distance they are from a querying user.

GCN architecture applied to semi-supervised classification tasks [9] shows the variety of tasks that GNN can solve, not only it can solve recommendation tasks but also classification.

Knowledge Graph Attention Network [13] is a GNN architecture that explicitly models the high-order connectivities in Knowledge Graphs in an end-to-end fashion. It recursively propagates the embeddings from a node's neighbors to refine the node's embedding and employs an attention mechanism to discriminate the importance of the neighbors.

Developing a GNN for a RS is currently still a challenging endeavor, as a balance between efficiency and accuracy needs to be met. Graph Convolutional Networks (GCNs) have already been proven to be efficient and highly scalable.

My M.Sc. project addresses the development and evaluation of GNN approaches, specifically envisioning applications in the recommendation of Gympass gyms. Taking inspiration from previous work such as PinSage [14], GNN at Decathlon [4] and LARS [10], I implemented a similar approach and evaluated it on Gympass data.

2 RELATED WORK

In this section, we will introduce the main related work regarding the project.

PinSage [14] is a random-walk Graph Convolutional Network that is highly scalable and capable of learning embeddings for nodes in web-scale graphs containing billions of objects. In other words, it can be said that PinSAGE greatly improved the scalability of GCNs by utilizing some key insights. Unlike the existing GCN algorithms that applied the form of multiplying the variable matrix using the entire Laplacian graph, PinSAGE used the method of sampling neighbors around the node and applying convolutions to it. Furthermore, it uses the CPU-bound producer to efficiently extract the neighbors around the node and prepare the variables needed for convolution. While iterative operations are minimized, the trained model can quickly generate embeddings for many nodes. There was also an innovation in the learning where the sampling of neighbors uses a small random walk technique where each node has an importance score, which is used in the Pooling/Aggregation stage. In the training, the PinSage method starts with simpler training sets but it provides more difficult training sets as the training advances.

A new recommender system approach based on GNN [4] by the Decathlon Research team to leverage all the available historical user data as well as interactions with user-item. Combining multiple data sources to build an efficient recommender system. The author create a graph for the model. The model has nodes as users, items and sports. Edges represent the interaction between a user and an item and could be of type click or type purchase, user, practicing, sport, in the way of a knowledge-graph enhanced graph. The basic embedding generation consists of a message-passing technique similar to the PinSage [14] algorithm, a spatial-based method. To generate recommendations, the author proposes a link prediction architecture, with models to predict the probability of an interaction between a user and an item. Using the user and item embeddings, the predicting function is a multilayer perceptron. To parametrize the model, the author proposes a max-margin loss function with negative sampling.

The authors investigate the utility of the knowledge graph (KG), which breaks down the independent interaction assumption by linking items with their attributes. The authors argue that in such a hybrid structure of KG and user-item graphs, high-order relations - which connect two items with one or multiple linked attributes - are an essential factor for successful recommendation. The solution found was Knowledge Graph Attention Network (KGAT) [13] which explicitly models the high-order connectivities in KG in an end-to-end fashion. It recursively propagates the embeddings from a node's neighbors (which can be users, items, or attributes) to refine the node's embedding, and employs an attention mechanism to discriminate the importance of the neighbors.

LARS [10] is a location-aware recommender system that uses location-based ratings to produce recommendations. Traditional recommender systems do not consider spatial properties of users nor items; LARS, on the other hand, supports a taxonomy of three novel classes of location-based ratings, namely, spatial ratings for non-spatial items, non-spatial ratings for spatial items, and spatial ratings for spatial items. LARS exploits user rating locations through user partitioning, a technique that influences recommendations with ratings spatially close to querying users in a manner that maximizes system scalability while not sacrificing recommendation quality. LARS exploits item locations using travel penalty, a technique that favors recommendation candidates closer in travel distance to querying users in a way that avoids exhaustive access to all spatial items. LARS can apply these techniques separately, or together, depending on the type of location-based rating available.

Experimental evidence using large-scale real-world reveals that LARS is efficient, scalable, and capable of producing recommendations twice as accurately compared to existing recommendation approaches.

Semi-supervised learning on graph-structured data [9] that is based on an efficient variant of convolutional neural networks which operate directly on graphs. The authors motivate the choice of convolutional architecture via a localized first-order approximation of spectral graph convolutions. The model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on multiple datasets, the authors demonstrate that their approach outperforms related methods by a significant margin.

3 RECOMMENDATION OF FITNESS VENUES USING GRAPH NEURAL NETWORKS

In this section, we introduce the model proposed. We will explain how the graph was built, how we designed the algorithm, how we trained the model, and handled the user location to give recommendations.

3.1 Model

The developed model builds upon the previous models PinSage [14] and GNN from Decathlon [4] by adapting them to the problem at hand.

Firstly, we built a tripartite graph with gyms, users, and activities, as we can see in Figure 1.

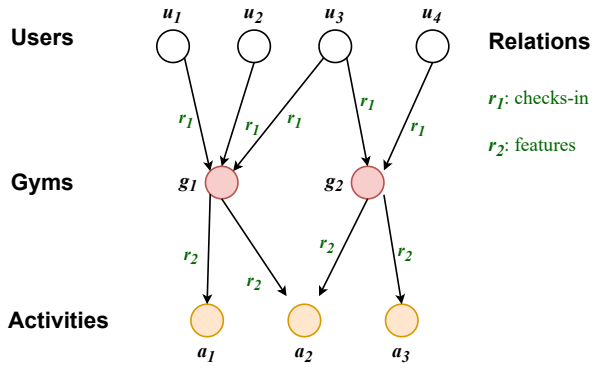


Figure 1: Tripartite graph with users, gyms, and activities

A gym and user are connected if the user had checked-in at least once in the gym. A gym and an activity are connected if the activity can be practiced in the gym. In the end, we have the following edges in the graph:

- user, checks-in, gym
- gym, checked-in-by, user
- activity, featured-by, gym
- gym, features, activity

We modeled the environment as a tripartite graph consisting of nodes in three disjoint sets, namely G (containing gyms), U (containing users), and A (containing activities). Consider V to be the node set of the full graph. The node features were the ones described in Section 4.2.

The task of generating an embedding z_u for each node u, which depends on the node's input features and the graph structure around this node is made through Algorithm .1.

The basic idea of Algorithm .1 is to transform the representations $z_v, \forall v \in \mathcal{N}(u)$ of u's neighbors, by reducing those representations into one by doing mean aggregation and multiply the result by a learnable weight matrix (Line 1 of Algorithm .1). This aggregation step provides a vector representation, n_u, of u's local neighborhood, $\mathcal{N}(u)$. Then, transform u's current representation z_u through a dense neural network layer, thereafter we sum the aggregated neighborhood vector n_u with the transformed u's current representation and pass the sum through a ReLU activation function (Line 2

Algorithm .1: Embedding Generation Layer

Input : Current embedding z_u for node u; set of neighbor embeddings $\{z_v | v \in \mathcal{N}(u)\}$

Output : New embedding z_u^{NEW} for node u

1 $\mathbf{n}_u \leftarrow \mathbf{Q} \cdot mean(\mathbf{z}_v | v \in \mathcal{N}(u))$;
2 $\mathbf{z}_u^{NEW} \leftarrow \mathsf{ReLU}(\mathbf{W} \cdot \mathbf{z}_u + \mathbf{n}_u)$;
3 $\mathbf{z}_u^{NEW} \leftarrow \mathbf{z}_u^{NEW} / ||\mathbf{z}_u^{NEW}||_2$

of Algorithm .1). The set of parameters of our model which we then learn is: the weight parameters ($Q^{(k)}$, $W^{(k)}$, $\forall k \in \{1, \cdots, K\}$). Furthermore, the normalization in Line 3 makes training more stable, and it is more efficient to perform an approximate nearest neighbor search algorithm for normalized embeddings. The output of the algorithm is a representation of u that incorporates both information about itself and its local graph neighborhood. Finally, we repeat the algorithm for as many layers as wished. The result is the output of the last layer.

The task's unique characteristics include the vast volume of data, which provides thousands of interactions for the model to be trained on. Such a huge graph cannot be fit on GPU utilization; batches are required. We require blocks that contain neighbors of all the nodes for which we want to construct embeddings in order to generate embeddings.

Our case includes data with thousands of interactions for the model to be trained on. Since a graph of this dimension cannot be fit on GPU utilization, batches are required. Batches include blocks that contain neighbors of all the nodes for which we want to construct embeddings.

Batching becomes more complicated as a result [4]. The model's layers go as deep as its building blocks. Each block layer contains every node needed to calculate the embeddings of the nodes in the layer below. Each batch of edges, therefore, contains blocks to build embeddings for each node connected by the edges, as well as a positive graph where the positive pairings are scored and a negative graph where the negative pairs are scored.

Consider that $\mathcal{L}$ is a set of labeled pairs of user and gym and $(u, g) \in \mathcal{L}$, where u is a user and g is a gym that the user checked-in, and thus corresponding to good recommendation candidate for the user. The model is trained in a supervised fashion using the max-margin loss function in Equation 1. The equation is based on the loss function by GNN at Decathlon [4]

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{E}} \sum_{v_n \in \mathcal{P}_{n,u}} \max\left(0, -f(z_u, z_v) + f(z_u, z_{v_n}) + \Delta\right) \quad (1)$$

where $\mathcal{E}$ is the set of edges on which training is done, $f(\cdot)$ is a cosine similarity function and $\mathcal{P}_{n,u}$ is a set of nodes negatively sampled from where v_n is drawn. The size of $\mathcal{P}_{n,u}$ and Δ are tunable hyperparameters.

The training process is intuitive in that we use positive pairs of instances as our training signal. The intention is for these positive pairs to receive higher scores from the model than the randomly generated negative pairs.

The full training loop of the model is very similar to one by the Decathlon team [4]. To summarize here is an overview of the

full training loop of the multiple training components presented throughout this section.

(1) Create the graph and divide the data into batches.
(2) For each batch, input the initial node features into the model.
(3) Each batch has its respective blocks. Each block corresponds to a model layer that will compute the updated representations of all the nodes in that block. The updated representations of the last layer are the final embeddings of all nodes.
(4) With all the final embeddings of the nodes in the batch, compute the loss.
    (a) For all positive edges, compute the similarity score between the user node and the item node.
    (b) For all negative edges, compute the similarity score between the user node and the item node.
    (c) For all negative edges, compute the similarity score between the user node and the item node.
    (d) The loss function is a max-margin loss. The positive score needs to be higher than the negative score by a predefined margin.
(5) Using the loss, parameterize the model. Compute the evaluation test metrics and use early stopping if MRR stops increasing for 10 successive epochs.

3.2 Scoring Function

Scoring is possible following the generation of the embeddings. The embedding of the edge's origin node, the user u, and the destination node, the gym g are inputs for the scoring function. The two embeddings are then compared using cosine similarity, represented as $P(u, g)$, yielding a score between 0 and 1.

When the model finishes the embedding generation, we first filter for each user the closest 100 gyms to their check-in and apply the ranking function which combines the user-gym embedding similarity and the euclidean distance between the user location and the gym location.

The ranking algorithm starts by running a 100-nearest-neighbor algorithm based on k-d tree [2] and euclidean distance between the user location at least 2 hours before the check-in and the gym location to populate the list R with 100 gyms with lowest euclidean distance.

With the 100 closest gyms, it computes the cosine similarity between them and the user. After that, it ranks each spatial item g for a querying user u based on $RecScore(u, g)$, inspired from the work of Location-Aware Recommender System [10], computed as:

$$RecScore(u, g) = P(u, g) - TravelPenalty(u, g), \quad (2)$$

$P(u, g)$ is the GNN recommendation model final embeddings cosine similarity of gym g with user u. $TravelPenalty(u, g)$ is the euclidean distance between u and i normalized to the same value range as $P(u, g)$.

Since the model failed to learn that recommending closer gyms to the other gyms a user has been are good recommendations, as we can see in Figure 2, and due to the user behavior, we decided to use the TravelPenalty algorithm [10] to solve the location problem.

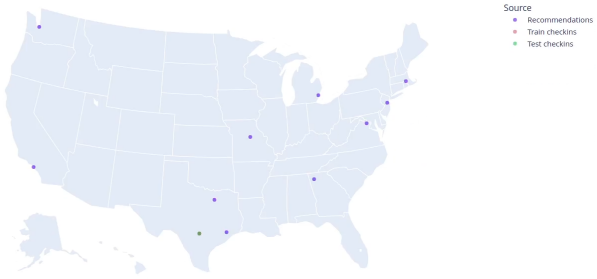
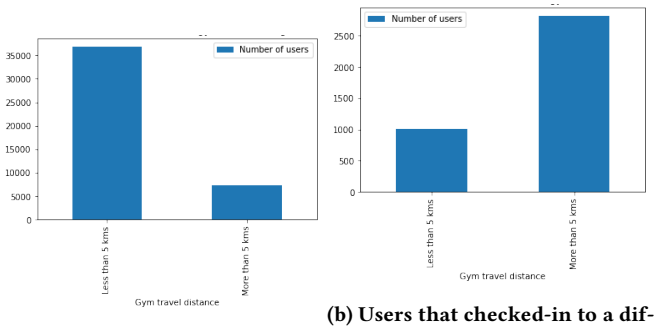


Figure 2: 10 recommendations for a user that went to the same gym in the training set and in the test set, using the RS without TravelPenalty. The RS recommendations are represented in blue, the gyms that the user checked-in in the training set are represented in brown, and the gyms that the user checked-in next, in the test set, are represented in green. The training gym check-in dot and the test gym check-in dot coincide.

The first version of the RS didn't include a ranking function that combined both cosine similarity and distance between the user and the gym as we have at the moment in the final version. The initial ranking function only included the cosine similarity between the user and the gym because we expected that the model was able to learn that recommending gyms closer to others that the user has been to would be good recommendations. As we can see in Figure 2, a user that just checked-ins in at a gym in San Antonio, Texas (green dot) would be recommended gyms very far away in Seattle and New York both at more than 2,500 kms from San Antonio.

Furthermore, since Gympass was interested in exploring recommendations for users that go to new gyms, we analyzed the data, as we can see in Figure 3, and we noticed that more than 70% of users that went to a different gym from the training set in test set traveled more than 5kms from their average check-in gym location. Considering all the users, we found that less than 20% of users traveled more than 5kms to check-in to a gym, we can see a significant difference between the traveled distance between all the users and the users that checked-in to different gyms in the test set where the latter tend to travel farther.

The TravelPenalty algorithm [10] applied in the re-ranking phase was the chosen method to solve the location problem. Since the TravelPenalty algorithm needs the user location, we used the user app location at least 2h before the check-in. We used the user location at least 2h before the check-in because, if the user location was very close in time to the check-in, the user location would match the gym location that the user was checking-in. Therefore if we used the user location at most 2h before the check-in we would be giving an unfair advantage to the recommendation system because it would only need to recommend the closest gym to maximize the offline metrics. Besides that, Gympass was interested in exploring the scenario where the user is just exploring the app searching for gyms nearby some hours before checking-in at a gym which supported using the user location 2h before the check-in.



(a) All users.

(b) Users that checked-in to a different gym in the test set.

Figure 3: Both charts show the mean users traveled distance between the average gyms location they checked-in in the training set and the gyms' location they checked-in in the test set. Left: Bar chart comparing the users that traveled less than 5 kms and more than 5kms to check-in to a gym. Right: Bar chart comparing the users that traveled less than 5 kms and more than 5kms to check-in to a different gym from training.

4 EXPERIMENTAL EVALUATION

This section describes the methods to train and evaluate the proposed model and describes the data and the data preprocessing applied to it.

4.1 Methodology

The initial stage of this thesis project focused on laying a good theoretical foundation that could support the reasoning for our proposed model. To this end, a survey of current state-of-the-art methods in the fields of GNN and RS was conducted. Special focus was given to work on products and sports since these were the main issues we attempt to address in Gympass.

Having understood what challenges GNN models faced, we set out to develop a RS based on GNN that is location aware in an attempt to ease some of the challenges affecting user query location and gyms' locations.

After downloading the Gympass dataset with US data which includes gym features, user-gym interactions, gym-activities edges, plans, and user locations, we cleaned and replaced text features with its BERT[5] text embeddings. Using temporal markers, the data is split into train and test sets. For all of the available check-ins, a defined period, i.e. from March 2021 to February 2022, is used for training, and the next month's time, i.e. from March 2022, is used for testing. We have two test sets, the test set and the test set only new check-ins where the test set only new check-in is a subset of the test set with only new check-ins between users and gyms to evaluate the model ability to recommend new gyms to users.

We build three baselines: one that only recommends the closest gyms based on the inferred user location, one which is a simpler model based on our proposed model, and another using only pre-processed embeddings to give recommendations.

The model is trained by first building the graph with edges and input node features and dividing the graph into batches of sampled graphs due to the large dimension of the full graph. For each batch, the embedding generation is done through message passing. With the final embedding layer, we compute the loss function to parameterize the model.

We evaluated this work with recommendation system metrics such as *Recall@k*, *MRR@k*, and *NDCG@k* at the cutoff point *k*.

The recommendation systems were implemented using the Python programming language, given that it allows for the quick and easy creation of different experiments, as well as the considerable machine learning and deep learning support it offers. Specifically, we took advantage of several libraries such as Pytorch[1], Deep Graph Library[2], PySpark[3], and MLFlow[4].

4.2 Data

We are using user US data to build the GNN model. The dataset includes gym features, user-gym interactions, gym-activities edges, plans, and user locations which are described in this section. In Table 1 we can see some statistics about the training and test data. Since most users check-in often in at the same gyms, it is interesting to evaluate how the RS performs when recommending gyms only to users who checked-in in at a different gym. For all of this, we extracted a subset of the test set with only new check-ins (i.e. check-ins that were not in the training set) to evaluate exactly that.

Table 1: Sets statistics

Statistic\Set	Train set	Test set	Test set only new check-ins
#users	12,865	877	217
#gyms	2,639	545	179
#activities	258	-	-
#check-ins	16,913	986	227
#gym-activities	14,854	-	-
Average user check-ins	1.315	1.124	1.024
Average gym check-ins	0.205	1.809	1.179
Average activities per gym	5.629	-	-
Average gyms per activity	57.574	-	-
#check-ins in non-train gyms	-	227 (23.02%)	227 (100%)
#users that checked-in in non-train gyms	-	217 (24.74%)	217 (100%)

4.2.1 Gym features. The gym features include two text fields: title and description, as we can see in Table 2. The title feature is the gym name, the description feature is a text written by the gym owner.

[1]https://pytorch.org/
[2]https://docs.dgl.ai/
[3]https://spark.apache.org/python/
[4]https://mlflow.org

We also have coordinate features such as latitude and longitude which mark the gym location. There are 2,639 gyms from different locations in the US. An example of gym textual features anonymized data is available in Table 2.

Table 2: Gym textual features

id	title	description
0	Crunch Fitness...	Why users love this gym?\nMembers love our gym...
1	Broadway Boxing Gym	What makes this place unique? \nWe have been a...
2	Lloyd Athletic Club	
3	Pilates Plus San Diego	Why users love this gym?\nWe provide unique in...

4.2.2 User-gym interactions. A user-gym interaction is when a user checks in at a gym at a given timestamp. This data is used to create edges between gyms and users. All check-ins made from March 7 2021 to March 3 2022 are fetched. Those interactions involve 2,639 different gyms and 12,865 users. An example of user-item interaction anonymized data is available in Table 3.

4.2.3 Gym-activities edges. A gym-activities edge is when a gym has an activity. This data is used to create edges between gyms and activities. Those edges involve 2,639 different gyms and 258 activities.

4.2.4 Plans. Each user and gym is associated with a value (max value and value, respectively) that corresponds to a plan. For example, if user u has a \$69.99 max value then he is in the plan Basic, as we can see from Table 4, and can only go to gyms inside plan Basic or below that plan. The same applies for the gyms but with the value. For example, if a gym g has a \$69.99 value then only users with plan Basic or higher, as we can see from Table 4, can check-in.

Users max value. The data is used to create engineered features. The dataset has 12,865 different users and their max value is associated with a certain starting and end date of that max value because users' plan value can change over time. A description of the user max value is available in Table 6.

Gyms value. The data is used to create engineered features. The dataset has 2,639 different gyms and their median value. A description of the gym value is available in Table 5.

4.2.5 Users' locations. The features include coordinate features which are latitude and longitude and a timestamp. There are 12,865 users. A description of the users' locations are available in Table 6.

Having understood the data being used, I will explain the data preparation and feature engineering applied to the data. Two main data preprocessing were applied: data preprocessing that uses the

Table 3: User-gyms check-ins

date	user_id	gym_id
2021-11-17 00:32:21.613	11307	1161
2022-02-04 13:05:30.946	6503	1161
2022-01-15 17:14:20.695	1931	1161
2021-12-17 13:03:26.346	10209	1161

Table 4: Plans' highest value

Order	Plan	Highest value/$
0	Starter	40
1	Basic	70
2	Bronze	100
3	Silver	150
4	Gold	250
5	Platinum	350
6	Diamond	450
7	Custom	1e+99

Table 5: Gyms features description

feature name	type	min	max	% nulls
latitude	float	19.644419	71.290174	0
longitude	float	-166.8080556	-68.7627325	0
value	float	9.99	449.99	0

Table 6: Users features description

feature name	type	min	max	% nulls
max_value	float	6.99	1999.0	0
valid_start_date	timestamp	2015-03-29 21:00:00	2022-07-20 01:00:00	0
valid_end_date	timestamp	2015-04-30 20:59:59	2022-08-22 00:59:59	0
latitude	float	25.7505585484564	47.74450538388441	0
longitude	float	-122.48305966157697	-70.9429543797924	0
timestamp	timestamp	2022-02-10 10:46:56.100	2022-03-09 23:57:32.690	0

activities embeddings and data preprocessing that uses the gym description.

4.3 Data Preprocessing That Uses the Gym Description

In this treatment, only the user and gym features are changed from the dataset described in Section 4.2. We start with the gym features, as we can see in Table 2. We keep all the other gym features but replace the text description with its text embedding, the embedding generation process is explained in more detail in Section 4.3.1. On the other hand, the user features are initiated with the average of the gym description embeddings that they checked-in at least once.

4.3.1 Text embedding. First, we applied multiple regex patterns to clean the text description field of gyms, we can see the rules in Table 7. Since we had some cases where the text description was left empty after the regex cleaning, in case the cleaned text description field was empty, we replaced it with the gym title. Afterwards,

Table 7: Regex rules for text description field

Rule	Example text	Cleaned text
HTML tags	important	important
HTML chars	Crossfit &#8594 rarr taekwondo &#x2192	Crossfit taekwondo
Only white spaces	\n \n	
Quotations	"One of the best gyms in LA"	One of the best gyms in LA
Ats	We are open @ the studio!	We are open the studio!
Hashtags	We are team #fitplus #gym4life	We are team
Repeated punctuation	One experimental class for free!!!!	One experimental class for free!

we generated text embeddings on the final gyms text description field using a BERT model [6] generating 768 floating point number vector.

4.4 Data Preprocessing That Uses the Activities Embeddings

After analyzing the gym text description, we found out that they were mostly marketing messages to attract customers into gyms which made them unhelpful because they missed essential useful information about the gyms that could help the model learn. To solve this issue, it was decided to replace the gym description embeddings with the average of their activities embeddings.

4.4.1 Text embedding. We generated text embeddings on the activities title field using a BERT model generating 768 floating point number vectors which replaced the activity title text field. The activity embeddings were reused to generate embeddings for the gyms. The gym embeddings were the average of activities embeddings associated with it.

4.4.2 Plans. So that the model could output embeddings that also took into account the plan information of the gym and user it was added new features to both.

Feature engineering. First we joined the gyms value and users max value, which are fields important to infer the user and gym plan, to generate the following new features. We applied an ordinal encoding with Gympass plan values, as we can see in Table 8, which generated a new field called plan_num. Then we created equal frequency bins with these values.

Table 8: Gympass user plans

Plan_num	Plan	Unlimited
0	Starter	40
1	Basic	70
2	Bronze	100
3	Silver	150
4	Gold	250
5	Platinum	350
6	Diamond	450
7	Custom	1e+99

4.5 Evaluation Metrics

Recommender systems are often assessed from either an online or an offline standpoint. Although offline approaches are the most common methods for evaluating recommender systems, online evaluation does offer often a true measure of the effectiveness of the system, mostly due to their viability and reproducibility in varied settings [1]. Precision and recall are common measurements.

The only user preferences that are recorded in a recommendation task with implicit data are those that are positive. Non-positive interactions don't always mean the user isn't interested; they might have just never seen the item before. The precision meter focuses on the accuracy of recommendations, which may include intriguing but rarely encountered items, whereas the recall metric concentrates on the positive interactions that really occurred. Therefore, recall should be utilized rather than precision.

Other popular metrics used in the literature are called ranking metrics such as Mean Reciprocal Rank (MRR) and Normalized Discounted Cumulative Gain (nDCG), which take the exponential decay of utility into account and suggest that "users are only interested in top-ranked items, and they do not pay much attention to lower-ranked items." [1]

The set of recommended items is denoted by $\mathcal{S}$ and let $\mathcal{G}$ represent the true set of relevant items (ground-truth positives) that are consumed by the user. The recall is computed according to the following equation:

$$Recall = \frac{|\mathcal{S} \cap \mathcal{G}|}{|\mathcal{S}|}. \tag{3}$$

We also evaluate the system using the Mean Reciprocal Rank (MRR), which takes into account the rank of the item j among recommended items for query u:

$$MRR = \frac{1}{m} \sum_{(u,j) \in L} \frac{1}{R_{u,j}}, \tag{4}$$

where $R_{u,j}$ is the rank of item j among recommended items for query u, and m is the total number of labeled item pairs.

We also use the NDCG which is computed with the discounted cumulative gain where the discount factor of item j is set to $\log_2(vj + 1)$, and v_j is the rank of item j in the test set I_u. Then, the discounted cumulative gain is defined as follows:

$$DCG = \frac{1}{m} \sum_{u=1}^{m} \sum_{j \in I_u} \frac{g_{uj}}{\log_2(v_j + 1)}. \tag{5}$$

In this case, the utility (or gain) of the user u in consuming item j is represented by g_{uj}. Typically, an exponential function of relevance (such as non-negative ratings or user hit rates) is specified as the value of g_{uj}:

$$g_{uj} = 2^{rel_{uj}} - 1. \tag{6}$$

Here, rel_{uj} is the ground-truth relevance of item j for user u, which is computed as a heuristic function of the ratings or hits. Then, the normalized discounted cumulative gain (NDCG) is defined as the ratio of the discounted cumulative gain to its ideal value, which is also referred to as the ideal discounted cumulative gain (IDCG).

$$NDCG = \frac{DCG}{IDCG}. \tag{7}$$

Repeating the calculation for DCG, but using the ground-truth rankings instead, yields the ideal discounted cumulative gain.

In the ranking segment, the resulting space is handled as an ordered set with a specific cut-off point k defined for each metric, comparing the top k ranked candidates of the RS with the top k ranked items.

In this work, we made use of several metrics to automatically evaluate the capability of the RS. Analyzing recommendations was performed through the computation of commonly used metrics in the recommendation systems domain, such as *Recall@k*, *MRR@k*, and *NDCG@k* at the cutoff point k. These metrics are calculated in the exact same manner as the previous equations, but only considering the top k retrieved candidates.

4.6 Results

To evaluate the model, we built 3 baselines: closest gym to user, only initial embeddings and bipartite model.

The baseline closest gym to user for each user outputs a recommendation list ordered by how far each gym is to the user starting from the closest to the farthest. The baseline was created due to the fact that US Gympass users have a preference for gyms close to them as we discovered that 75% of users travel less than 5kms to a gym, as we can in Figure 3.

The baseline only initial embeddings uses the initial user and gym features described in the data preprocessing described in Section 4.3 and for each user and gym, the cosine similarity ranking function is applied so that given a query user u, returns a gyms list whose embeddings are most similar to the query user's embedding. The list is ordered by how similar the item embedding is to the query u. The baseline was created because Gympass recommendation systems using only embeddings were successfully built and deployed with high evaluation metrics.

The baseline bipartite model is based on the model built in Section 3.1. We built a simple graph that consisted of a bipartite graph with gyms and users connected if the user had checked-in at least once in the gym. In the end, we have the following graph edges:

- user, checks-in, gym
- gym, checked-in-by, user

We modeled the environment as a bipartite graph consisting of nodes in two disjoint sets, namely G (containing gyms) and U (containing users). Consider V to be the node set of the full graph. The node features were the ones described in the data preprocessing that uses the gym description, as described in Section 4.3. Since the model needs to have the same feature size for gyms and features, the data preprocessing that uses the activities can not be used. After the model generates the users and gyms embeddings, the cosine similarity ranking function is applied so that given a query user u, returns a gyms list whose embeddings are most similar to the query user's embedding. The list is ordered by how similar the item embedding is to the query u. We applied an efficient similarity search library called Faiss [8]. Faiss contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM.

Table 9 presents the effects of the different proposed preprocessing techniques and recommendation systems on the recommendation task for the different test sets.

When it comes to recommending users with gyms in *only new check-ins in test set*, it seems that the tripartite model improves every metric very significantly by at least 2.95 times over the second best baseline metrics.

Since the RS *closest gyms to user* only uses the coordinates features from both users and gyms which do not change in both data preprocessing, the RS will have the same values on the metrics for both data preprocessing, as we can see in Table 9. The RS *closest gyms to user* seems to be the best in terms of recall for both data preprocessing using the *normal test set* but has the lowest metrics in MRR and NDCG. This could mean that the RS *closest gyms to user* could be interesting in scenarios where we are recommending to all users that it is more important to just retrieve relevant gyms than it is to have the most relevant gyms right on top of the recommendations. The RS *closest gyms to user* seems to have relatively low metrics when being evaluated against the *only new check-ins test set* having the lowest MRR and NDCG. The RS *closest gyms to user* relatively low recall value seems to be explained by the majority of check-ins not being in the first 10 closest gyms from the inferred user location. This could mean that these users usually go beyond just the first 10 gyms near them.

The RS *only initial bipartite model embeddings* and *bipartite model* do not have metrics for the data processing that uses the activities because that data preprocessing has users and gyms features with different dimensions and, since this recommendation systems have the limitation of only accepting the same dimension of features for gyms and users, they can not generate embeddings of the features with the data preprocessing that uses the activities. However, since in the data preprocessing that uses the gym description both users and gyms have the same feature dimension, the RS can compute metrics. For the data preprocessing that uses the gym description and *normal test set* the RS *only initial bipartite model embeddings* seems to have the highest metrics in terms of MRR and NDCG and the second best recall metric. However, for the *only new check-ins test set* it seems to have one of the lowest metrics. The difference in values between the *normal test set* and *only new check-ins test set* could be explained by the RS using user embeddings initialized with the average of the gyms they checked-in on the training set. Since most users repeat gyms in the training set on the test set, it might allow the RS to have high metrics in *normal test set*. However, since in the *only new check-ins test set* we only have check-ins of users to new gyms they didn't go to in the training set, it fails to generalize to these new check-ins.

The RS *bipartite model* in the data preprocessing that uses the gym description and *normal test set* seems to have the second best metrics in terms of MRR and NDCG and in the *only new check-ins test set* seems to have the second highest overall metrics but by a very significant difference. Since the RS *bipartite model* is initialized with the same embeddings used in the RS *only initial bipartite model embeddings*, might start with a tendency to recommend gyms in training. This behavior might be aggravated by the fact that the *bipartite model* architecture shares the same GCN layer and their parameters for both the check-in and checked-in-by edge which makes the model overfit the training data leading to poor metrics when trying to recommend new gyms to users, as we can see in Table 9.

The RS tripartite model using the data preprocessing that uses the activities seems to have the highest MRR and NDCG for the *normal test set* by more than 2.1 times. Using the same data preprocessing but with the *only new check-ins test set* the *tripartite model* seems to have the highest metrics by at least more than 2.74 times over the second best. Using the data preprocessing that uses the gym description and *normal test set*, the RS seems to have significantly lower metrics than the two RS with the highest metrics. Using the *only new check-ins test set* and the same data preprocessing, it seems that the *tripartite model* improves every metric very significantly by at least 2.95 times over the second best metrics. The fact that the tripartite model seems to have the best metrics might show how well the model is able to generalize training data to new gyms that the user didn't go yet. The *tripartite model* is able to leverage

Table 9: Recall, MRR, and NDCG for data preprocessing that uses the activities and embeddings and data preprocessing that uses the gym description for the normal test set and the only new check-ins test set.

Data preprocessing	Data preprocessing that uses the activities						Data preprocessing that uses the gym description					
Recommendation system\Test set	Normal test set			Only new check-ins test set			Normal test set			Only new check-ins test set		
	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10
Closest gyms to user	**0.782**	0.129	0.269	0.170	0.032	0.061	**0.782**	0.129	0.269	0.170	0.032	0.061
Only initial bipartite model embeddings	-	-	-	-	-	-	0.657	**0.417**	**0.471**	0.119	0.049	0.063
Bipartite model	-	-	-	-	-	-	0.599	0.346	0.403	0.119	0.057	0.069
Tripartite model	0.711	**0.538**	**0.565**	**0.467**	**0.198**	**0.257**	0.591	0.299	0.361	**0.502**	**0.234**	**0.294**

both the content information of the gyms, users, and activities, and the relations between each other to generate embeddings for both users and gyms. The metrics seem to show that the embedding of a user, that is going to a new gym, is similar to the embedding of the new gym going next which might lead the gym to rank higher on the user recommendations and increase the RS metrics.

The hyperparameters combination that had the highest overall evaluation metrics was the following for the Tripartite Model:

- learning rate: $[0.0001, 0.0008]$
- delta: $[0.26, 0.30]$
- hidden embeddings dimension: 2^7
- output embeddings dimension: 2^7
- number of layers: 3
- negative sample size: $[1200, 1500]$

5 CONCLUSIONS AND FUTURE WORK

My M.Sc. project aimed to understand if a GNN model can recommend new Gympass gyms to users.

The results of our GNN RS seem promising for the case of recommending users to new gyms that they are visiting for the first time. They seem to show that a RS based on DL can predict which new gym a user will go to next better than only location based recommendation systems or simpler GNN models.

The obtained results support the understanding that a DL model can recommend new Gympass gyms to users. The main contribution of this work relies on building and validating a RS based on GNN that infers how to model Gympass complex environment into a graph, using a GNN model architecture learns users' past behaviors and with the ranking function recommends gyms to users. This thesis provides a GNN recommendation system with a trained model showing promising results compared to the baselines.

For future work, it could be interesting to extend the experiments reported in this dissertation to also other Gympass products such as classes and apps. It would also be interesting to add more data about the gyms to the gym features.

Besides BERT embeddings, there are other text embedding pretrained models that could be used in this thesis for comparison since BERT only supports English but Gympass has gym descriptions available in multiple languages.

Gympass gyms also have pictures that could be used in the RS so that it takes into account the quality of the gym pictures when recommending them to users.

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems - The Textbook*. Springer, 1–498. ISBN: 978-3-319-29659-3.

[2] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18, 9, (Sept. 1975), 509–517. DOI: 10.1145/361002.361007.

[3] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (RecSys '16). Association for Computing Machinery, Boston, Massachusetts, USA, 191–198. ISBN: 9781450340359. DOI: 10.1145/2959100.2959190.

[4] Jérémi DeBlois-Beaucage. 2021. Advanced recommender systems for e-commerce: graph neural networks at decathlon. (2021).

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. http://arxiv.org/abs/1810.04805 arXiv: 1810.04805.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. http://arxiv.org/abs/1810.04805 arXiv: 1810.04805.

[7] M. Gori, G. Monfardini, and F. Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. (2017). arXiv: 1702.08734 [cs.CV].

[9] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. (2016). DOI: 10.48550/ARXIV.1609.02907.

[10] Justin J. Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F. Mokbel. 2012. Lars: a location-aware recommender system. In *2012 IEEE 28th International Conference on Data Engineering*, 450–461. DOI: 10.1109/ICDE.2012.54.

[11] A. Sperduti and A. Starita. 1997. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8, 3, 714–735. DOI: 10.1109/72.572108.

[12] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*. C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, (Eds.) Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf.

[13] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, (July 2019). DOI: 10.1145/3292500.3330989.

[14] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, (July 2018). DOI: 10.1145/3219819.3219890.

[15] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '16). Association for Computing Machinery, San Francisco, California, USA, 353–362. ISBN: 9781450342322. DOI: 10.1145/2939672.2939673.

[16] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: a survey and new perspectives. *ACM Comput. Surv.*, 52, 1, Article 5, (Feb. 2019), 38 pages. DOI: 10.1145/3285029.

[17] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '17). Association for Computing Machinery, Halifax, NS, Canada, 635–644. ISBN: 9781450348874. DOI: 10.1145/3097983.3098063.

[18] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2021. Graph neural networks: a review of methods and applications. (2021). arXiv: 1812.08434 [cs.LG].