CS 545 (Fall 2022) Machine Learning
Computer Science Department
Colorado State University

[Dec 12, 2022]

Project Report: Restaurant Recommender System based on Machine Learning with Graphs

Ronaldo Armando Canizales Turcios
Sushmitha Kasimsetty Ramesh

Word count: 4078.

Abstract

How can an AI model learn to predict users' ratings of restaurants? The answer to this question is key to creating a recommender system, and that's what our project is about. We use two approaches to obtain the inputs to feed-forward neural networks: (1) feature engineering uses node-level, and edge-level features, which means restaurant characteristics and user preferences, and (2) graph representation learning uses task-independent strategies to capture graph structure into embedding vectors. This way, we predicted edges (rating values) on a bipartite graph; demonstrating not only that machine learning models can be used with graph data, but also that graph embeddings can obtain similar or better results than hand-crafted feature vectors.

1 Introduction

One of the most appealing characteristics of machine learning is its ability to solve problems, such as regression or classification, using task-independent techniques like artificial neural networks. The data can be related to economics, medicine, education, geoscience, etc.

However, data scientists should be familiar with the topic because, most of the time, features must be hand-crafted after careful data wrangling (cleaning, filtering, etc.). This approach is named "feature engineering" and usually takes considerable time and effort. Such a task becomes more complicated when the data is structured in complex ways, such as images, text, or graphs.

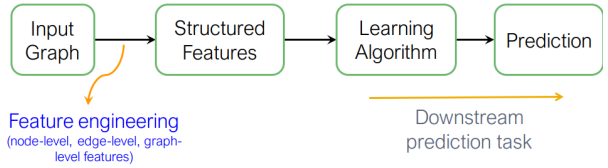


Figure 1: Feature engineering approach

On the other hand, "representation learning" takes the "task-independent" idea to the next level. Alleviating the need to do feature engineering every single time. The goal is to automatically generate features that will be used as inputs for a machine learning algorithm (like a neural network).

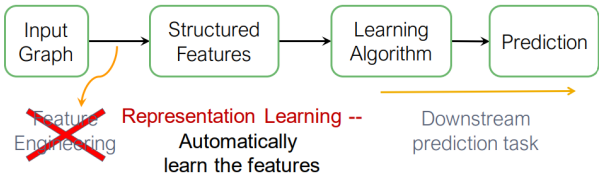


Figure 2: Representation learning approach

Graph representation learning techniques can calculate embedding vectors for nodes, edges, subgraphs, or entire graphs. This project uses random-walk-based algorithms to generate vectors that capture the relationship between restaurants and users. Then, a neural network is trained to map such vectors to ratings.

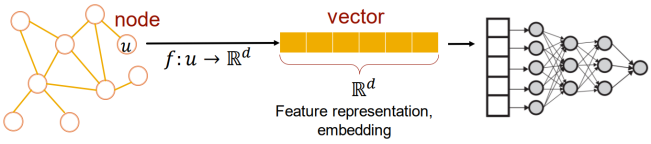


Figure 3: Machine Learning with Graphs basic idea

Generally speaking, four task types can be solved with machine learning with graphs: node-level, edge-level, subgraph-level, and graph-level. Some examples are node classification, link prediction, node clustering, graph classification, generation, and evolution.

Our project focuses on link prediction, that is, predicting new links based on existing ones. The key is to design good features for a pair of nodes.

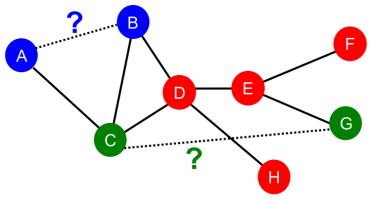


Figure 4: Link-Level Prediction Task

According to Leskovec (2022), a bipartite graph can be used to represent interactions (e.g., purchases) between users and items. For our purposes, interactions will represent ratings between users and restaurants.

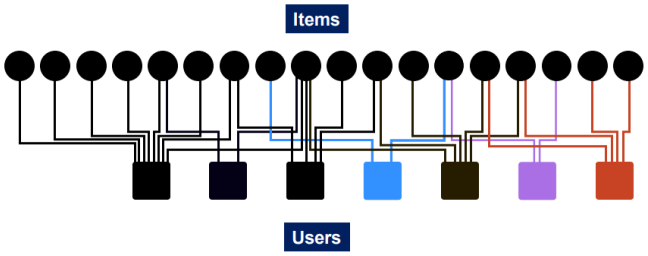


Figure 5: Bipartite User-Item Graph

Our task is to apply both approaches (feature engineering and representation learning) to the bipartite graph and obtain a vector for each node, which will be appended to obtain vectors for each edge (user-restaurant).

Then, we have to divide the vectors set into three subsets: train, validation, and test. Finally, train a neural network and aim to predict the missing edges. This approach is called "links missing at random".

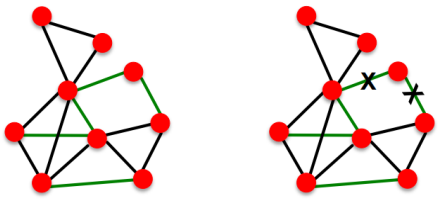


Figure 6: Links missing at random

We used the publicly available dataset from Medellin and Serna (2012) that contains nine files: restaurant general data, payment methods, cuisines, schedules, parking availability, consumers general data, preferred cuisines, payment methods, and ratings.

The authors used semantic rules like the ones shown in the following Figure to estimate ratings. Our project takes an entirely different approach to the same goal.

user - service profile
$\text{person}(X) \wedge \text{hasOccupation}(X, student) \wedge$ $\text{restaurant}(R) \wedge \text{hasCost}(R, low) \rightarrow \text{select}(X, R)$
user - environment profile
$\text{person}(X) \wedge \text{isJapanese}(X, true) \wedge$ $\text{queryPlace}(X, USA) \wedge \text{restaurant}(R) \wedge$ $\text{isVeryClose}(R, true) \rightarrow \text{select}(X, R)$
environment - service profile
$\text{currentWeather}(today, rainy) \wedge \text{restaurant}(R) \wedge$ $\text{space}(R, closed) \rightarrow \text{select}(R)$
Relations
$\text{likesFood}(X, Y)$ X: person, Y: cuisine-type
$\text{currentWeather}(X, Y)$ X: query, Y: weather
$\text{space}(X, Y)$ X: restaurant, Y: $\{closed, open\}$

Figure 7: Some semantic rules and relations used in Vargas-Govea et al. (2011) to predict ratings

Our project consists of several files, summarized in the following dependencies graph. There are 3 file types: jupyter notebooks, python files, and "pickle" files.

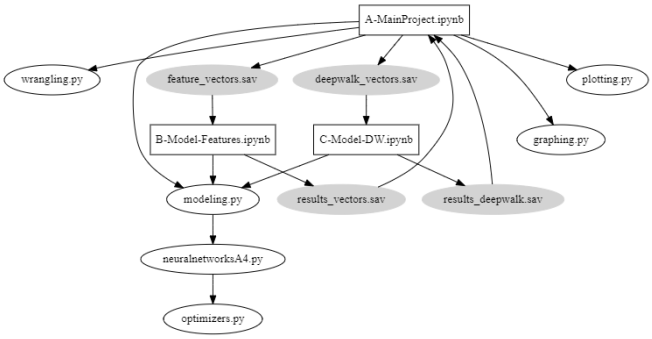


Figure 8: Our project's files and dependencies

The most important file is the jupyter notebook "A-MainProject" because it contains the whole flow of our work. First, it **loads the raw data** from the nine files and putting all together into three dictionaries (restaurants, users, and ratings). Second, it **depurates**, encodes, replaces, and renames the data with the help of the file wrangling.py. Third, it **separates** the data into the train (76%), validation (12%), and test (12%) datasets.

Fourth, it performs **feature engineering** and saves the resulting vectors in the file feature_vectors.sav in order to "B-Model-Features" to read them and experiment with 100+ hyperparameters configurations. This allows to keep "A-MainProject" simple, nice-looking (avoids long output logs), and fast to execute; also, "B-Model-Features" could be executed in a more powerful computer and store the results into the file "results_vectors," which will be finally read back by "A-MainProject."

The same trick is played with "C-Model-DW" through the intermediate files "deepwalk_vectors.sav" and "results_deepwalk"; after using the **EasyGraph library** to calculate the **graph representation learning** embedding algorithm DeepWalk within the file "graphing.py."

Fifth, it shows the **confusion matrix** and data likelihood plot per each target variable: ratings, food ratings, and service ratings, alongside the best ten hyperparameters configurations found. It uses the file "modeling.py."

Finally, it uses the **Graphviz library to plot** some nice-looking Figures that the reader will find in this report, with the help of the file "plotting.py."

```
def obtain_inputs(r, u):
    vector = []
    # relation restaurant-user ------------------------------
    vector.append(distance(r['latitude'], r['longitude'],
                           u['latitude'], u['longitude']))
    vector.append(coincidences(r['cuisines'], u['cuisines']))
    vector.append(coincidences(r['payments'], u['payments']))
    vector.append(comparison(r['ambience'], u['ambience']))
    vector.append(r['price'])
    vector.append(u['budget'])
    vector.append(r['parking_availability'])
    vector.append(u['car_owner'])
    vector.append(r['alcohol_friendliness'])
    vector.append(u['alcohol_friendliness'])
    vector.append(r['smoking_friendliness'])
    vector.append(u['smoking_friendliness'])
    vector.append(r['dress_code'])
    vector.append(u['dress_preference'])
    # just restaurant ---------------------------------
    vector.append(r['weekly_hours'])
    vector.append(r['accessibility'])
    vector.append(r['days'])
    vector.append(r['franchise'])
    vector.append(r['other_services'])
    vector.append(r['outdoor'])
    # just user ---------------------------------------
    vector.append(u['age'])
    vector.append(u['salary'])
    vector.append(u['single'])
    vector.append(u['kids'])
    return vector
```

Figure 9: Feature engineering vectors.

2 Methods

2.1 Feature engineering

According to Leskovec (2022), link-level features can be obtained by concatenating, averaging, adding, multiplying, subtracting, or calculating the distance between the features of the two nodes involved.

This results in one vector per each edge; each vector has 24 elements that can be grouped into the logical subsets:

Relation restaurant-user: distance, preferred cuisines coincidence, preferred payment method coincidence, preferred ambiance coincidence, price & budget, parking availability & car ownership, alcohol friendliness, smoking friendliness, and dress code preference.

Restaurant related: weekly hours available, days open, accessibility, franchise, extra services, and outdoorsy.

User related: age, salary, singleness, and child custody.

After that, 885 vectors were used to train several artificial neural networks. Then, the best ten hyperparameter configurations were chosen by their validation accuracy. Finally, a confusion matrix is obtained from the 138 vectors of the test dataset.

2.2 Graph representation learning

According to Leskovec (2022), graph representation learning is a way to learn node and graph embeddings for downstream tasks without feature engineering.

The goal is to encode nodes, so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph.

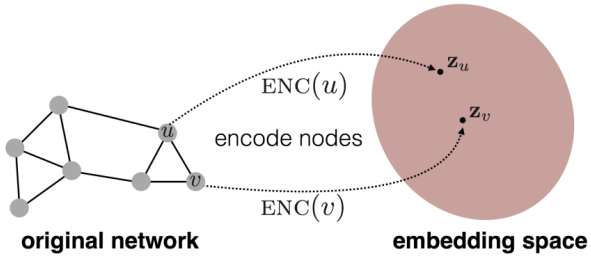


Figure 10: Embedding nodes

Thus, we need two functions: an encoder that maps from nodes to embeddings and a decoder that maps from embeddings to the similarity score. The most straightforward approach is "shallow encoding", where encoding is just an embedding lookup into a matrix, and decoding uses a dot product between node embeddings.

It is important to note that this is an unsupervised way of learning node embeddings because node and edge features are not used. The goal is to estimate the embeddings based on the network structure directly.

This technique is task-independent; embeddings are not trained for a specific task but can be used for any task.

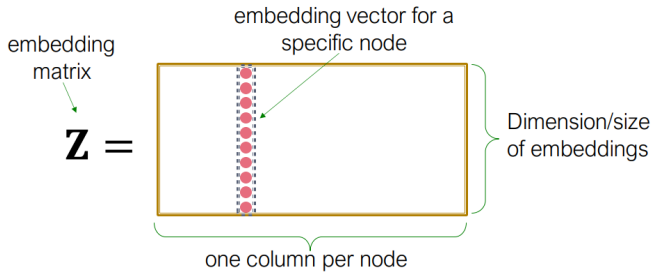


Figure 11: Shallow encoding

The main idea is that each node is assigned a unique embedding vector. There are many methods for measuring the proximity of nodes. For instance, shortest path (green, blue, and purple), common neighbors (orange), and Random-Walks-based approaches (red).

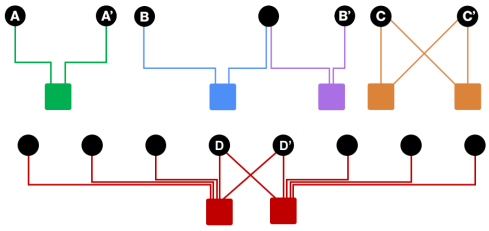


Figure 12: Node proximity measurements

A **Random Walk** on a graph is the stochastic sequence of points visited through the following process. If a graph and a starting point are taken, we move to a neighbor selected at random, then to a neighbor of this point at random, and so on.

Random-Walk Embeddings aims to estimate the probability of visiting node v on a random walk starting from node u using some random walk strategy R, and optimize such embeddings to encode these random walk statistics.

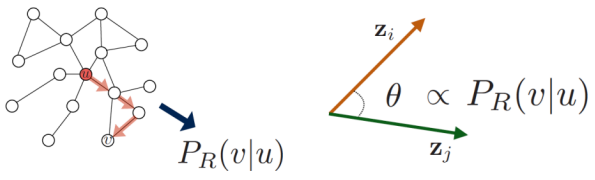


Figure 13: Random-Walks Embeddings

The advantages to this approach are two. First, it allows taking into account both local and higher-order neighborhood information (structure of the graph). Second, there is no need to consider all edges when training, only those pairs that co-occur on random walks. This is particularly useful in large or densely connected graphs.

There are many possible ways in which Random Walks can be applied. The most simple idea, DeepWalk, was proposed by Perozzi et al. (2014) just to run fixed-length, unbiased random walks starting from each node.

According to the survey Goyal and Ferrara (2018), node2vec performs better on node classification, while other methods perform better on link prediction. Random walk approaches are generally more efficient.

Applying Random Walks on a bipartite graph can be interpreted as quantifying the number of visits of neighboring nodes by random walks started at a node Q.

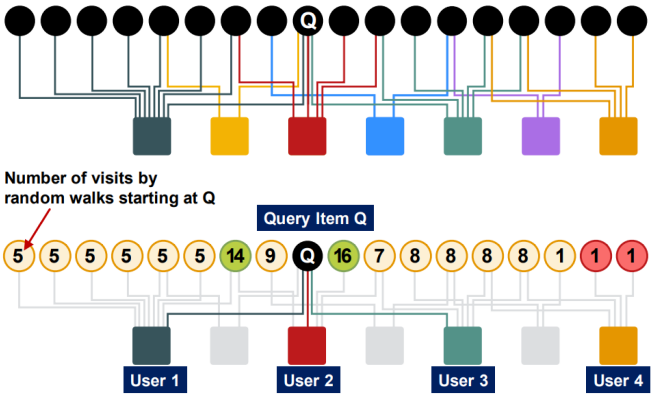


Figure 14: Random Walks on bipartite graphs

In our project, this means finding the restaurants that a particular user is more likely to visit and vice-versa (the users that are most likely to visit a given restaurant).

We used the **Easy Graph** "Optimized Graph Data Processing Library" Systems and Group (2022), publicly available for the Python programming language. Easy Graph has implemented several methods of graph embeddings, like DeepWalk Perozzi et al. (2014), Node2Vec Grover and Leskovec (2016), and others.

Furthermore, we calculated the Graph embedding matrix via DeepWalk for the bipartite graph and obtained a vector of length 128 for each node, either restaurant or user. According to Leskovec (2022), link-level features can be obtained by concatenating node embeddings, thus, resulting in a vector of length 256 for each edge (interaction between restaurants and users).

Doing this in Python is straightforward; use the built-in function extend to concatenate lists instead of append for adding individual elements into a list. Analogously to the vectors obtained by feature engineering, 885 vectors were used to train several neural networks in the annexed jupyter notebook.

```
def obtain_inputs(embeddings, r, u):
    vector = []
    vector.extend(embeddings[r])
    vector.extend(embeddings[u])
    return vector
```

Figure 15: Representation learning vectors.

It is important to remember that separate notebooks offer the possibility to execute "B-Model-Features" and "C-Model-DW" in a more powerful computer and store the results in intermediate files.

In addition to that, the notebook "A-MainProject" can remain free of long strings resulting from training 200+ hyperparameters configurations. And the confusion matrices can be plotted without re-training any model.

3 Results

3.1 Data Wrangling

The dataset Medellin and Serna (2012) contains more than one thousand ratings. Each user has at least three ratings, which is very convenient because we can include at least one rating from each user in all sub-datasets (train, validation, and test).

Using the library Systems and Group (2022) allows us to plot the graph in a very straightforward way, unfortunately resulting in a crowded graphical representation.

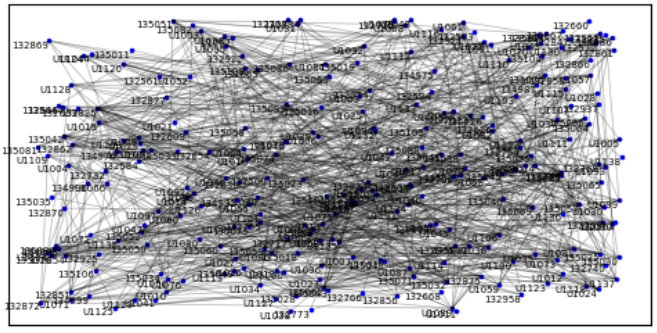


Figure 16: Data plot (using EasyGraphs library)

The first file loaded was *geoplaces2.csv*, which contains the following fields for each restaurant: place ID, latitude, longitude, the_geom_meter, name, address, city, state, country, fax, zip, alcohol, smoking_area, dress_code, accessibility, price, URL, Rambience, franchise, area, and other_services. Some of them are self-explanatory, and some others do not.

After that, the files *chefmozaccepts.csv*, *chefmozcuisine.csv*, *chefmozparking.csv*, and *chefmozhours4.csv* were loaded and appended to the dictionary restaurants. Those files contain the fields: Rpayment, Rcuisine, parking_lot, hours, and days, respectively.


```
('135034', {'placeID': '135034', 'latitude': '22.140517', 'longitude': '-101.021422', 'the_geom_meter': '0101000020957F000026D92BB4894858C161A7552DA2B04B41', 'name': 'Michiko Restaurant Japones', 'address': 'Cordillera de Los Alpes 160 Lomas 2 Seccion', 'city': 'San Luis Potosi', 'state': 'SLP', 'country': 'Mexico', 'fax': '?', 'zip': '78210', 'alcohol': 'No_Alcohol_Served', 'smoking_area': 'none', 'dress_code': 'informal', 'accessibility': 'no_accessibility', 'price': 'medium', 'url': '?', 'Rambience': 'familiar', 'franchise': 'f', 'area': 'closed', 'other_services': 'none', 'Rpayment': ' cash VISA MasterCard-Eurocard American_Express', 'Rcuisine': ' Japanese', 'parking_lot': ' none', 'hours': ' 13:30-23:00; 13:30-23:00; 13:30-19:00;', 'days': ' Mon;Tue;Wed;Thu;Fri; Sat; Sun;'})
```


Figure 17: Restaurant nodes' raw data

After that, three depurating actions were performed on the restaurant data. First, drop those fields that do not contribute to the goal of classifying the ratings; maybe because there is so much missing data (fax, zip, and URL), the data is the same across the entire dataset (country), or it is irrelevant to the study (name). Second, encode some fields from nominal values to numeric ones, for example, one-hot encoding (franchise, area) or scales (accessibility, other services). Third, replacements like transforming the daily schedule for each restaurant into the weekly available hours and days.


```
('132717', {'latitude': '23.7318602', 'longitude': '-99.1504365', 'dress_code': -1, 'accessibility': 0, 'price': 1, 'franchise': 0, 'other_services': 0, 'days': 7, 'alcohol_friendliness': -1, 'smoking_friendliness': -1, 'outdoor': 0, 'ambience': 'familiar', 'payments': ' cash', 'cuisines': ' Fast_Food', 'parking_availability': 0, 'weekly_hours': 84.0})
```


Figure 18: Restaurant nodes' depurated data

Next, the file *userprofile.csv* was loaded, which contains the majority of the user's personal information: user ID, latitude, longitude, smoker, drink_level, dress_preference, ambience, transport, marital_status, hijos (kids), birth_year, interest, personality, religion, activity, color, weight, budget, and height.

After that, the files *userpayment.csv* and *usercuisine.csv* were loaded in order to append the fields Upayment and Ucuisine to the dictionary users.

('U1016', {'userID': 'U1016', 'latitude': '22.156247', 'longitude': '-100.977402', 'smoker': 'false', 'drink_level': 'casual drinker', 'dress_preference': 'informal', 'ambience': 'friends', 'transport': 'on foot', 'marital_status': 'single', 'hijos': 'independent', 'birth_year': '1991', 'interest': 'eco-friendly', 'personality': 'thrifty-protector', 'religion': 'Catholic', 'activity': 'student', 'color': 'green', 'weight': '70', 'budget': 'medium', 'height': '1.67', 'Upayment': ' cash', 'Rcuisine': ' Cafe-Coffee_Shop Contemporary Regional Fusion Japanese Portuguese American Indian-Pakistani Eastern_European Lebanese Moroccan Barbecue Polynesian Polish'})

Figure 19: User nodes' raw data

Next, the same three depurating actions were performed on the user's data. Dropped fields are those considered irrelevant to user rating judgment (religion, favorite color, weight, and height). Some encoded fields are similar to restaurants (smoker, drink_level, dress_preference), so they will be used as relations in further steps. Some special replacements were made because of the unequal distribution of the values (kids, marital status), and others due to the nature of the field (from birth year to age).

Finally, some fields were renamed to make both datasets (restaurants and users) more consistent.

('U1109', {'latitude': '22.303308', 'longitude': '-101.05468', 'dress_preference': 1, 'ambience': 'quiet', 'budget': 0, 'single': 1, 'kids': 0, 'age': 33, 'smoking_friendliness': -1, 'alcohol_friendliness': -1, 'car_owner': 0, 'salary': 0, 'payments': ' cash', 'cuisines': ' Mexican'})

Figure 20: User nodes' depurated data

The last file loaded was *rating_final.csv*, which contains the ratings made per each user to different restaurants. It is important to mention that each restaurant was rated at least once, and each user rated at least three restaurants.

{'userID': 'U1133', 'placeID': '132766', 'rating': '1', 'food_rating': '1', 'service_rating': '0'}

Figure 21: Ratings edges' data raw.

Three different ratings were performed: overall rating, food rating, and service rating. All ratings were always performed; there is no instance of a "partial rating."

3.2 Model A: feature engineering based

As explained in the previous section, the feature vector for each edge (representing a rating given by a user to a restaurant) has a length of 24.

```
TRAIN - Inputs size:(885, 24)   Outputs size:(885, 1)(885, 1)(885, 1)
VAL   - Inputs size:(138, 24)   Outputs size:(138, 1)(138, 1)(138, 1)
TEST  - Inputs size:(138, 24)   Outputs size:(138, 1)(138, 1)(138, 1)
```

Figure 22: Feature engineering vectors' size.

After performing experiments for more than one hundred different hyperparameter configurations, the best classifiers' confusion matrices are presented as follows.

	0	1	2
0	43.3	20.0	36.7
1	24.5	38.8	36.7
2	25.4	27.1	47.5

	0	1	2
0	40.6	25.0	34.4
1	20.5	41.0	38.5
2	16.4	28.4	55.2

	0	1	2
0	37.2	32.6	30.2
1	14.9	55.3	29.8
2	12.5	39.6	47.9

Figure 23: Confusion matrices using features: rating (left), food rating (center), and service rating (right)

Overall rating classification. Accuracies obtained: training 73.44%, validation 47.82%, and testing 43.47%. According to the confusion matrix, whose values are percentages of classified instances, it is easier for the model to predict ratings of 0 and 2 than values of 1.

Food rating classification. Accuracies obtained: train 77.40%, validation 50.72%, and test 47.82%. The confusion matrix shows an increase in the prediction accuracy, reaching more than 55% in ratings of value 2 (the highest value, meaning maximum user satisfaction) and slight changes in other values.

Service rating classification. Accuracies obtained: train 75.70%, validation 43.47%, and test 47.10%. The confusion matrix shows lower accuracy in value 0 (maximum dissatisfaction of the user) but an increase of above 55% in values of 1.

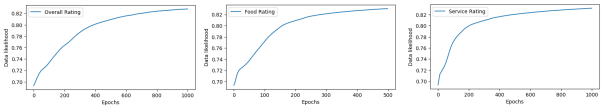


Figure 24: Data likelihood across training epochs

All classifiers present some level of overfitting. The data likelihood obtained by the three models is close to 0.82.

The hyperparameters configurations tested are detailed as follows. Neural network hidden layer structures [[], [10], [15, 15], [5, 10, 5], [10, 15, 10]]. Optimizing methods: [sgd, adam, scg]. Training epochs: [500, 1000, 2000]. Learning rates: [0.005, 0.01, 0.05].

The following table shows that, at least for overall rating modeling, there is no preferred hyperparameter value in any criteria; both simple and complex structures obtained reasonably good results. The same happens with epochs and learning rates.

	Structure	Method	Epochs	Learning Rate	Training Acc	Validation Acc	Test Acc
103	[10, 15, 10]	scg	1000	-	97.514124	48.550725	42.028986
92	[10, 15, 10]	sgd	2000	0.05	76.610169	48.550725	42.028986
75	[5, 10, 5]	adam	1000	0.005	73.446328	47.826087	43.478261
29	[10]	sgd	2000	0.05	70.395480	47.826087	43.478261
60	[15, 15]	scg	500	-	97.966102	47.101449	39.130435
44	[15, 15]	sgd	500	0.05	53.220339	47.101449	48.550725
0	[]	sgd	500	0.005	51.751412	46.376812	47.826087
74	[5, 10, 5]	adam	500	0.05	74.463277	46.376812	44.927536
71	[5, 10, 5]	sgd	2000	0.05	63.615819	46.376812	53.623188
46	[15, 15]	sgd	1000	0.01	51.525424	46.376812	44.202899

Figure 25: Best ten hyper-parameter configurations found (out of 135 tested) based on validation accuracy

A similar scenario occurs with the classification of food ratings. With the exception of that, more complex structures gave higher training accuracies and sightly high validation and testing accuracies. There are three cases in which 90%+ values were obtained for the structure [10, 15, 10], most probably due to overfitting.

	Structure	Method	Epochs	Learning Rate	Training Acc	Validation Acc	Test Acc
80	[5, 10, 5]	adam	2000	0.05	72.429379	53.623188	45.652174
58	[15, 15]	adam	2000	0.01	87.118644	51.449275	43.478261
31	[10]	adam	500	0.01	77.401130	50.724638	47.826087
104	[10, 15, 10]	scg	2000	-	98.418079	50.000000	44.202899
47	[15, 15]	sgd	1000	0.05	71.977401	49.275362	44.927536
86	[10, 15, 10]	sgd	500	0.05	49.265537	49.275362	49.275362
65	[5, 10, 5]	sgd	500	0.05	49.378531	49.275362	48.550725
96	[10, 15, 10]	adam	1000	0.005	97.062147	48.550725	44.202899
21	[10]	sgd	500	0.005	48.587571	48.550725	45.652174
97	[10, 15, 10]	adam	1000	0.01	96.384181	48.550725	41.304348

Figure 26: Best ten results on food rating classification

Regarding service ratings, the only difference is that higher values of training epochs are present in the best ten results. Also, lower learning rates are less frequent.

	Structure	Method	Epochs	Learning Rate	Training Acc	Validation Acc	Test Acc
92	[10, 15, 10]	sgd	2000	0.05	76.497175	47.826087	45.652174
47	[15, 15]	sgd	1000	0.05	70.621469	45.652174	39.130435
33	[10]	adam	1000	0.005	78.418079	44.202899	36.956522
34	[10]	adam	1000	0.01	75.706215	43.478261	47.101449
88	[10, 15, 10]	sgd	1000	0.01	46.327684	43.478261	42.753623
41	[10]	scg	2000	-	79.435028	43.478261	44.927536
104	[10, 15, 10]	scg	2000	-	98.418079	42.753623	42.028986
102	[10, 15, 10]	scg	500	-	95.593220	42.753623	40.579710
101	[10, 15, 10]	adam	2000	0.05	77.740113	42.753623	44.927536
91	[10, 15, 10]	sgd	2000	0.01	51.751412	42.753623	44.927536

Figure 27: Best ten results on service rating classif.

3.3 Model B: graph repres. learning based

Our second model will try to get similar or better results using information only from the graph's structure (and not from the node's features), which means how the restaurants and users interact, which restaurants are the most popular, and which restaurants are popular among similar users (as explained in the previous section, with the measurements of node proximity such as shortest path, common neighbors, and Random Walks).

In order to perform the same experiment as with the previous model, the data has been divided into the same three subsets (train, validation, and test) using the same proportions (76%, 12%, and 12%).

The key difference is that the Random-Walk-based algorithm DeepWalk Perozzi et al. (2014) has been used to generate the node embeddings (of length 128) that will be concatenated as suggested by Leskovec (2022) to obtain edge embeddings of length 256.

```
TRAIN - Inputs size:(885, 256)  Outputs size:(885, 1)(885, 1)(885, 1)
VAL   - Inputs size:(138, 256)  Outputs size:(138, 1)(138, 1)(138, 1)
TEST  - Inputs size:(138, 256)  Outputs size:(138, 1)(138, 1)(138, 1)
```

Figure 28: Representation learning vectors' size.

This approach's name is "links missing at random" and has been used by Medellin and Serna (2012) with **two important differences.** First, there is no complete randomness in picking the links to be subtracted from the train set; **the idea is to subtract two random ratings per user.**

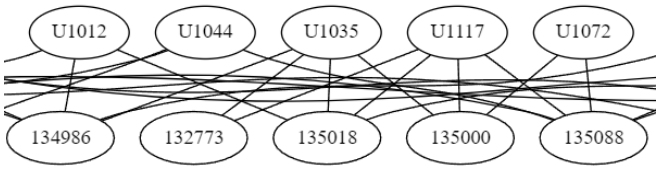


Figure 29: Sample training instances (885 in total, 76%)

Doing this allows each user to have at least one rating in the train set, thus, avoiding having any disconnected nodes. Each user will be connected to a restaurant, which will be connected to other users, allowing the algorithm to find preferences and communities. This is possible because each user in the original dataset rated at least three restaurants.

The second difference is that **we use a validation set,** which is avoided in Medellin and Serna (2012).

The validation set is used to sort the models based on their validation accuracy. As seen in the following Figure, the validation set is more sparse than the training set, but it contains at least one rating per user. Thus, allowing us to compare models comprehensively.

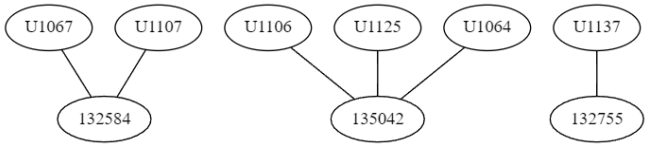


Figure 30: Sample validat. instances (138 in total, 12%)

Finally, the test set is very similar to the validation test, as they share the same characteristics. The confusion matrices are calculated from the test data.

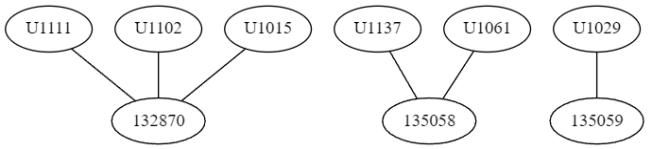
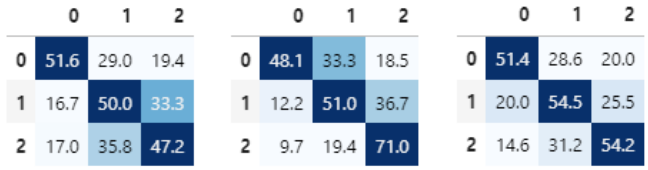


Figure 31: Sample test instances (138 in total, 12%)

After performing experiments for more than one hundred different hyperparameter configurations, the best classifiers' confusion matrices are presented as follows.



	0	1	2
0	51.6	29.0	19.4
1	16.7	50.0	33.3
2	17.0	35.8	47.2

	0	1	2
0	48.1	33.3	18.5
1	12.2	51.0	36.7
2	9.7	19.4	71.0

	0	1	2
0	51.4	28.6	20.0
1	20.0	54.5	25.5
2	14.6	31.2	54.2

Figure 32: Confusion matrices using embeddings: rating (left), food rating (center), and service rating (right)

Overall rating classification. Accuracies obtained: training 77.74%, validation 57.24%, and testing 49.27%. Outperforming rating prediction for values 0-1; and obtaining similar accuracy for value 2, the reader can see that graph representation learning is valuable. Also, the misclassifications done by the model are primarily between values 1-2, and fewer were done between values 0-2 as in the previous model.

Food rating classification. Accuracies obtained: train 77.51%, validation 57.24%, and test 59.42%. The confusion matrix shows the best results obtained so far. It is important to highlight three points. First, a rating value of 2 (maximum user satisfaction) is classified with 70%+ accuracy. Second, mistakes are made mainly between values 0-1 and 1-2, decreasing the severity of the error. Third, mistakes between values 0-2 are almost half the previous model.

Service rating classification. Accuracies obtained: train 100%, validation 56.52%, and test 53.62%. The only confusion matrix with 50%+ in all the values in the main diagonal and the remaining values almost evenly distributed among other cells.

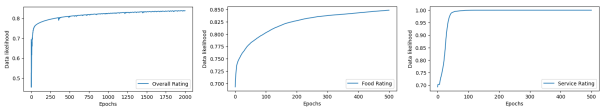


Figure 33: Data likelihood across training epochs

All classifiers present some level of overfitting, even though the proportion between parameter amounts and input size is lower than in the previous model. Data likelihood varies as follows: 0.8+ (overall rating), 0.85+ (food rating), and approximately 1.0 (service rating).

The hyperparameters configurations tested are very similar to the ones used previously, with the exemption of hidden layer structures, that has been increased due to the approximately **ten-fold increase in the input size** (256 now against 24 before). Neural network hidden layer structures [[], [50], [100, 100], [200, 150, 100]]. Optimizing methods: [sgd, adam, scg]. Training epochs: [500, 1000, 2000]. Learning rates: [0.005, 0.01, 0.05].

Unexpectedly, there is a clear pattern in overall rating classification. Simple structures outperformed more complex ones; the Adam optimizing method was also preferred, as shown in the following Figure. On the other hand, learning rates and epochs seem to vary greatly.

	Structure	Method	Epochs	Learning Rate	Training Acc	Validation Acc	Test Acc
25	[50]	sgd	1000	0.01	77.062147	57.971014	39.855072
16	[]	adam	2000	0.01	78.079096	57.971014	46.376812
15	[]	adam	2000	0.005	77.966102	57.971014	47.101449
11	[]	adam	500	0.05	74.124294	57.971014	44.927536
14	[]	adam	1000	0.05	76.158192	57.246377	46.376812
12	[]	adam	1000	0.005	74.689266	57.246377	47.101449
17	[]	adam	2000	0.05	77.740113	57.246377	49.275362
13	[]	adam	1000	0.01	75.932203	57.246377	47.826087
18	[]	scg	500	-	79.661017	56.521739	47.101449
9	[]	adam	500	0.005	73.672316	56.521739	44.927536

Figure 34: Best ten hyper-parameter configurations found (out of 108 tested) based on validation accuracy

Opposite to the previous classifier, the target variable "food rating" preferred more complex hidden layer structures, for instance [200, 150, 100]. The downside is that more overfitting is present among the best results. That's why we picked the configuration #18 that has 57% and 59% for validation and test accuracies, respectively. Another curious insight is that neither the optimizing method, epochs amount, nor learning rate seems to influence obtaining good results.

	Structure	Method	Epochs	Learning Rate	Training Acc	Validation Acc	Test Acc
50	[100, 100]	sgd	2000	0.05	100.000000	57.971014	56.521739
65	[200, 150, 100]	sgd	500	0.05	100.000000	57.246377	50.000000
18	[]	scg	500	-	77.514124	57.246377	59.420290
61	[100, 100]	scg	1000	-	100.000000	56.521739	48.550725
33	[50]	adam	1000	0.005	100.000000	56.521739	50.724638
71	[200, 150, 100]	sgd	2000	0.05	100.000000	56.521739	50.000000
73	[200, 150, 100]	adam	500	0.01	100.000000	56.521739	50.000000
83	[200, 150, 100]	scg	2000	-	100.000000	55.797101	53.623188
55	[100, 100]	adam	1000	0.01	100.000000	55.797101	50.000000
29	[50]	sgd	2000	0.05	100.000000	55.797101	57.246377

Figure 35: Best ten results on food rating classification

Last but not least, a fairly simple hyperparameter configuration was chosen as the best for classifying service rating: one hidden layer of 50 units, 500 epochs, and the optimizer method SCG. Chosen because it gave the best validation accuracy and more than 50% accuracy per class. Similarly to previous experiments, neither hidden layer structure, optimizing method, epochs amount, nor learning rates seem to influence overall accuracy.

It is important to note that better results were obtained, with the downside that overfitting was also more common among these results.

	Structure	Method	Epochs	Learning Rate	Training Acc	Validation Acc	Test Acc
39	[50]	scg	500	-	100.000000	56.521739	53.623188
29	[50]	sgd	2000	0.05	100.000000	55.797101	47.826087
51	[100, 100]	adam	500	0.005	100.000000	55.072464	47.826087
83	[200, 150, 100]	scg	2000	-	100.000000	54.347826	39.855072
70	[200, 150, 100]	sgd	2000	0.01	100.000000	54.347826	49.275362
54	[100, 100]	adam	1000	0.005	100.000000	54.347826	43.478261
52	[100, 100]	adam	500	0.01	100.000000	54.347826	43.478261
41	[50]	scg	2000	-	100.000000	54.347826	46.376812
25	[50]	sgd	1000	0.01	75.706215	53.623188	47.101449
58	[100, 100]	adam	2000	0.01	100.000000	53.623188	43.478261

Figure 36: Best ten results on service rating classif.

3.4 Comparison with state-of-art

In order to compare our work with the state-of-art, the paper Li and Chen (2012) will be briefly discussed. The researchers stated that "To capture users' interests better and make effective recommendations, it is necessary to combine multiple models and make effective use of different types of data, such as user information, item information, and transaction information (business transactions, browsing activities, review activities, etc.)."

According to the authors, there are generally two types of recommendation tasks, predicting purchase vs. predicting rating. We will focus on the rating-related task.

Our methodology shares **similarities** with theirs, like using a bipartite graph, extracting graph structure through random-walks-based techniques, and using node features. Also, there are **differences** between both methodologies, like using an artificial neural network vs. a software vector machine, datasets size (ours has 138 users, 130 restaurants, and 1,161 ratings while theirs has 278,858 users, 271,379 books, and 1,149,780 ratings), and mixing structural and features information vs. using them separately due to time constraints (1 month).

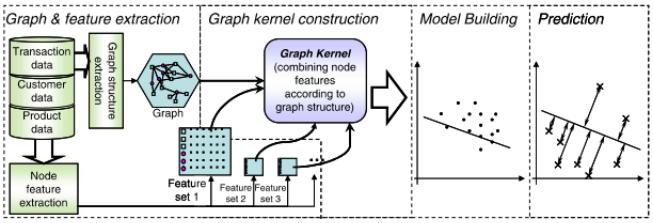


Figure 37: A graph kernel-based recommendation framework used by Li and Chen (2012)

Nevertheless, both results are comparable and demonstrate that our approach, although not in the range of hundreds of thousands of instances, captures the essence of a recommender system and its algorithms.

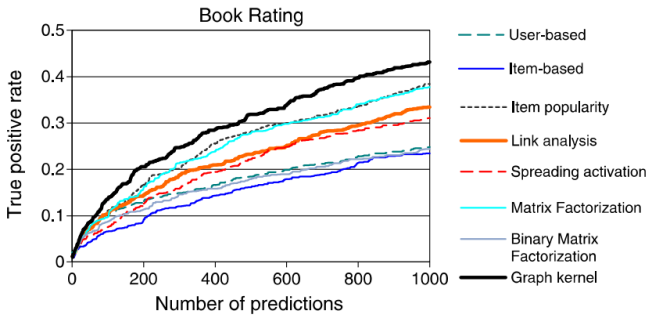


Figure 38: ROC curves of top 100 recommendations

4 Conclusions

In our project, we present a machine-learning-with-graphs-based approach for recommendations viewed as a classification of users' ratings on restaurants. We modeled the recommendation task as an edge prediction problem in a bipartite graph.

We learned not only about feature engineering on graph data but also about graph representation learning techniques, such as encoding nodes and edges into an embedding space that approximates similarity in the graph.

We learned how to develop neural networks that are much more broadly applicable, a new frontier beyond sequences or image data. Graphs are complex, have arbitrary size, complex topological structures, no fixed node ordering, contain multimodal features, and are usually more dynamic than other data structures. We familiarized ourselves with scientific writing, state-of-art algorithms, and open-source libraries and datasets.

One of the most challenging steps was the data wrangling of the users, restaurants, and rating information. Python was an invaluable tool and also were the libraries Easy Graphs and Graphviz Ellson et al. (2022).

It was a rewarding surprise that graph representation learning gave better results than the traditional feature engineering approach. We look forward to including this exciting technique in our toolkit for future work.

4.1 Future Work

We would like to explore three sub-approaches. First, combine graph structural features (learned through more complex random-walks-based techniques) with note information to fully exploit the ability of machine learning to solve graph-related tasks. Second, perform feature selection to find which node-level feature combinations lead to better results and avoid possible dimensionality problems. Third, explore Folded/Projected Bipartite Graphs Leskovec (2022) to take advantage of user-user and restaurant-restaurant relations.

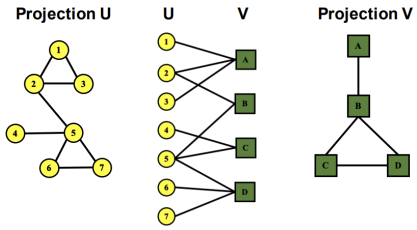


Figure 39: Folded/Projected Bipartite Graphs

References

John Ellson, Emden Gansner, Yifan Hu, and Stephen North. Graphviz: open source graph visualization software. The Graphviz Authors, 2022. URL https://graphviz.org/.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 2018. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2018.03.022. URL http://www.sciencedirect.com/science/article/pii/S0950705118301540.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016. URL https://arxiv.org/abs/1607.00653.

Jure Leskovec. Cs224w: Machine learning with graphs, 2022. URL https://web.stanford.edu/class/cs224w/.

Xin Li and Hsinchun Chen. Recommendation as link prediction: A graph kernel-based machine learning approach. In *Proceedings of the 9th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '09, page 213–216, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781605583228. doi: 10.1145/1555400.1555433. URL https://doi.org/10.1145/1555400.1555433.

Rafael Medellin and Juan Serna. Restaurant and consumer data. UCI Machine Learning Repository, 2012. URL https://archive.ics.uci.edu/ml/datasets/Restaurant%20&%20consumer%20data.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, aug 2014. doi: 10.1145/2623330.2623732. URL https://doi.org/10.1145%2F2623330.2623732.

Mobile Systems and Networking Group. Easygraph optimized graph data processing library. Fudan University, 2022. URL https://easy-graph.github.io/docs/index.html.

Blanca Vargas-Govea, Gabriel Gonzalez-Serna, and Rafael Ponce-Medellin. Effects of relevant contextual features in the performance of a restaurant recommender system. 2011. URL https://ceur-ws.org/Vol-791/paper8.pdf.