Predicting the star rating of a business on Yelp using graph convolutional neural networks

Ana-Maria Istrate
Department of Computer Science
Stanford University
aistrate@stanford.edu

Abstract

Social media platforms have been rising steadily in recent years, influencing consumer spaces as a whole and individual users alike. Users also have the power of influencing the popularity of businesses or products on these platforms, driving the success level of different entities. Hence, understanding users' behavior is useful for businesses that want to cater to users' needs and know what market segment to direct efforts towards. In this paper, we are looking at how the star rating of a business on Yelp is determined by the profile of users who have rated it with a high score on Yelp. We are defining a graph between users on Yelp and businesses they gave high ratings to, and using graph convolutional neural networks to find node embeddings for businesses, by aggregating information from the users they are connected to. We show how a business's star rating can be predicted by aggregating local information about a business's neighborhood in the Yelp graph, as well as information about the business itself.

1 Introduction

Social media platforms have become prevalent in recent years, making it easier for users to engage with other people, as well as give and get feedback on services, businesses and products. Yelp, in particular, gathers people interested in food-related services, businesses most of which include restaurants. People have a chance to write reviews and give businesses a star rating from 1 to 5. We are looking into how the profiles of users who like a certain business are influencing the star rating of that business. Knowing this information could help businesses better cater their needs to specific categories of users, or know what types of user profiles they should direct their marketing efforts towards. In tackling this problem, we are using graph convolutional neural networks to compute embeddings for nodes in the Yelp graph, which is determined by users and businesses, connected by edges if a user gave a high star rating to a particular business. Graph convolutional neural networks (GCN) is a method that applies a convolution around a node to gather that node's neighbors' information and combine it with its own information. In the end, the learned convolutions are applied on nodes in order to compute node embeddings. The embeddings can then be used as input for node classification. In our case, we are

looking to classify a given business into one of the star-rating categories.

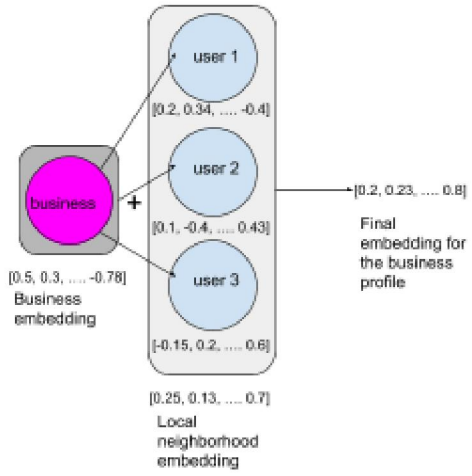

Figure 1. Basic convolution around a business node

We show that simple information about a user's profile can lead to meaningful embeddings for users and businesses alike, and that graph convolutional neural networks are an exciting area of research in the field of understanding and modeling consumer profiles and behavior.

2 Benefits of GCNs

Graph convolutional neural networks have been shown to give good results on link prediction and node classifications tasks ([1], [3]). One of the main benefits of GCNs is that there is a lot parameter sharing: more shallow approaches usually train one unique embedding vector for each node, which means that the number of parameters grows linearly with the number of nodes in the graph. Moreover, most other approaches that compute node embeddings (Node2Vec [4], DeepWalk [5]) are transductive, which means that they can only generate embeddings for nodes seen doing training. Hence, these methods require retraining every time that new nodes are added to the graph. Especially in a graph defining a social media platform, similar to Yelp, where users are being added daily, this is unfeasible, as training can be expensive. In contrast, GCNs generalize very well and are inductive, meaning that they can compute embeddings for nodes that have not been seen during training by simply applying the aggregator functions.

3 Relevant Work

Related papers are in the field of graph convolutional neural networks. One of the first papers to introduce graph convolutional neural networks is *Semi-supervised Classification With Graph Convolutional Neural Networks*, where Kipf et al. show the success of GCNs on the node classification task for Cora and Pubmed datasets. They provide a semi-supervised approach using a graph convolutional neural network using a localized first-order approximation of spectral graph convolutions. It starts by computing a matrix $\hat{A} = D^{-1/2}AD^{-1/2}$, where A is an adjacency matrix. The model is then defined by:

$$Z = f(X, A) = softmax(\hat{A}\ Relu(\hat{A}XW^{(0)})W^{(1)})$$

where $W^{(0)}$ and $W^{(1)}$ are learned matrices. It uses a semi-supervised log loss. The method proposed in the paper is mainly applicable to small graphs, as it needs to know the entire

Laplacian during training. In fact, this is one of its main weakness, that it cannot be applied to graphs that are large in size or constantly increasing, as it needs to operate on the entire Laplacian during training, which could be expensive.

In *Inductive Representation Learning on Large Graphs,* Hamilton et al. provide a different approach to defining the convolution on graphs than [1]. While Kipf et al. define the aggregation by a two-layer neural network using a Relu, followed by a Softmax, this paper defines a number of aggregator functions that learn to aggregate information from a different number of steps away from a given node. In fact, this is one of the main strengths of the paper, which compares different types of aggregator functions. For instance, the mean aggregator just averages information from local neighborhoods, while the LSTM aggregator is able to operate on a random permutation of the node's neighbors, despite not being symmetric. Moreover, the pooling aggregator performs a max-pooling on each neighbor's vector after it is being fed through a fully-connected neural network.

Another strength of this paper is that it leverages node features, showing how they can improve performance, in comparison with [1], where graphs were not as feature rich. The paper also introduces random walks on the graph as a way of getting positive samples and uses negative-sampling.

This method can be used with both an unsupervised and supervised log-loss function:

$$L = -\log(\sigma(z_u^T z_v)) - Q \cdot E_{vn - Pn(v)} \log(\sigma(-z_u^T z_{vn}))$$

where v = node that co-occurs near u on a random walk
Pn = distribution of negative samples
At test time it is simply applying the learned aggregator functions to get embeddings for new nodes.

While successful on small datasets, applying GCNs on large scale datasets has still been challenging. In one of the most recent papers in the field, *Graph Convolutional Neural Networks for Web-Scale Recommender Systems,* Ying et al. successfully apply GCNs to compute embeddings for nodes in the Pinterest graph, which contains billions of pins. This is the most recent paper in the field, and its biggest contribution is that it is working with a really large graph, containing 3 billion nodes and 18 billion edges (the Pinterest graph). They compute node embeddings using GCNs and then provide recommendations via nearest neighbors search in the embedding space. It is the first paper to show that graph convolutional neural networks can be leveraged on web-scale graphs. Architecturally, it is very similar to GraphSage, the model proposed in [2], improving upon it by adding engineering artifices to address the scale of the problem and algorithmic contributions for better performance.

In terms of engineering improvements, they propose a producer-consumer architecture where they use the CPU and GPU resources efficiently for different types of computations. For instance, they use the CPU to sample node network neighborhoods, get the node features, store the adjacency list, reindex and perform negative sampling, and the GPU to run the training, running one GPU computation at a iteration and a CPU computation at the next iteration in parallel.

They also do on-the-fly convolutions, where they sample a neighborhood around a node and dynamically construct a computation graph from the sampleed neighborhood, meaning that they alleviate the need to operate on the entire graph during training, a shortcoming of the previous two approaches.

They also have a MapReduce pipeline to minimize re-computation of the same nodes' embeddings. In contrast with [2], they use an importance pooling aggregator, where they weigh the importance of node features. They define neighborhoods by sampling the computation graphs with random walks around a node. Another contribution of the paper is introducing curriculum training, where the algorithm is fed harder and harder examples during training, in order to learn to differentiate better.

4 Model Architecture

In this section, we present the model architecture.

4.1 Graph definition

We define the following graph $G = (V, E)$:

$V = \{u \in Set_{users},\ b \in Set_{businesses}\}$

$E = \{(u, b)$ if user u gave business b at least with a 3.5 rating$\}$

By using this definition for E, we are creating a graph containing businesses and clients who gave them high ratings. We are essentially assuming that a client who rated a business with a high score is more likely to resemble this business profile in the embedding space, and provide more meaningful information in the neighbor aggregation phase.

4.2 Node features

Each entry in the graph, business or user, contains some associated information, which we leverage as input features to the model. These will be the inputs to the graph convolutional neural model. The features we end up using are the following:

For a business:

$x_v^0 =$ {neighborhood, city, state, postal_code, latitude, longitude, review_count, alcohol, bike_parking , accepts_credit_cards, caters, drivethru, goodforkids, hastv, noise_level , outdoor_seating, restuarants_price_range, delivery, goodforgroups, pricerange, reservations, table_service, takeout, wifi}

And for a user

x_v^0 = {useful, funny, cool, #fans, average_star, compliment_hot, compliment_more, compliment_profile, compliment_cute, compliment_list, compliment_note, compliment_plain, compliment_cool, compliment_funny, compliment_writer, compliment_photos}

Some of these features are transformed into categorical features, while some are continuous. At the end of the input feature extraction, each business ends up having a feature vector of size 24, and each user ends up having a feature vector of size 16.

4.3 Models

In this section, we present the models we experimented with.

4.3.1 Multi-class Logistic Regression

As a baseline, we are using a simple multi-class logistic regression model on the business's features. In this model, we are not using the graph structure or the users' information at all. We use the cross-entropy loss function

4.3.2 Linear Regression

As another baseline, we are also using linear regression on the business' node features. We use the mean-squared loss function.

4.3.2 Graph Convolutional Neural Networks

We are combining the information about a business's local neighborhood together with the embedding of the business itself and pass it through a neural network in order to predict a final star rating. Essentially, we are modeling a business's profile by combining information both about the business itself and the profile of the users that like this business, getting the latter by applying a convolution around the users connected to that business in the graph.

We use a 1-layer graph convolutional neural network, following the definition from GraphSage [2]., where we use as input the features described in 4.2. For each node, we average the signals from all neighbors (we do not perform any sampling). Then, we concatenate the result with the embedding of the node at the current layer and pass the result through a neural network. Basically, for each business node x_v, we start with an input feature x_v^0 as given in 4.2, and then at each layer, we compute:

$$x_v^k = Relu(W_k[\frac{\sum_{u \in N(v)} x_u^{k-1}}{N(v)}, x_v^{k-1}]),\ k > 0$$

where x_v^i = v's embedding in layer i

The output of our model is the learned matrice W_k, which can then be applied to

any node in order to get an embedding using the above equation.

We are only using 1-layer.

4.4 Prediction

For each of the business nodes in the graph, we predict its star-rating. For both multi-class logistic regression and GCNs, we consider the possible cases:

1. Predicting one of 9 possible ratings: $[1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]$
2. Predicting one of 5 possible ratings: $[1, 2, 3, 4, 5]$

For linear regression, we predict a continuous score, and then round either up or down, depending on whether the predicted value x is smaller than or greater than the floor of that value + 0.5. We use the cross-entropy loss function for both logistic regression and GCNs.

6 Data

We are using part of the Yelp dataset, made available at https://www.yelp.com/dataset as part of a challenged proposed by Yelp. The dataset which contains ~6 million reviews, ~200k businesses and ~280k pictures, covering 10 metropolitan areas and 2 countries. We are only considering businesses that have at least one review and users that gave at least one review. After performing other minor dataset cleaning operations, we are left with 146526 businesses and 1518169 users. The data is split 90% into train and 10% into train. Out of the training data, 10% is used for validation

7 Evaluation

For evaluation, we are using the accuracy as a metric:

$$accuracy = \frac{\#correct\ star\ ratings}{\#all\ star\ ratings}$$

8 Results

	Training accuracy	**Test accuracy**
Logistic Regression, 9 classes	0.25	0.254
GCN, 9 classes	0.267	0.254
Logistic Regression, 5 classes	0.383	0.3912
GCN, 5 classes	0.39	0.4
Linear Regression	0.2	0.021

Logistic regression was trained for 1000 epochs, linear regression for 10000 epochs, and GCNs for 50 epochs (because they are significantly slower than the other two methods). All models used an Adam optimizer and were implemented in pytorch. Learning rate for logistic regression was 0.001, and for GCNs 0.1.

Training graphs can be seen below:

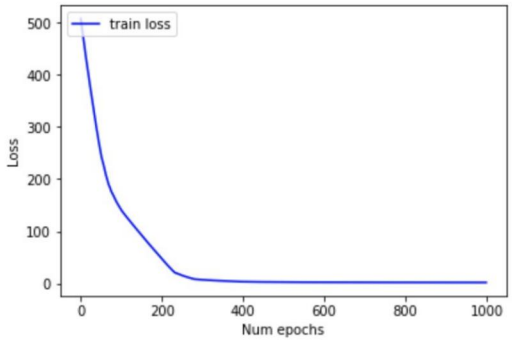

Figure 2. Train loss for logistic regression, 9 classes

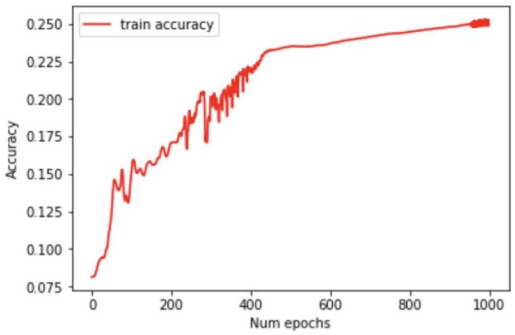

Figure 3. Training accuracy for logistic regression, 9 classes

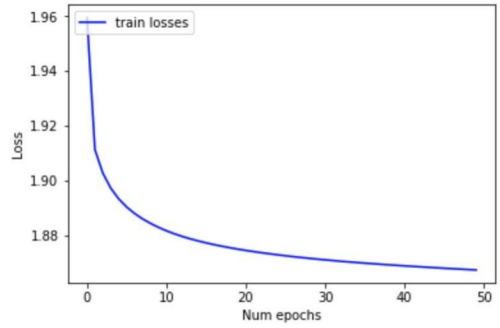

Figure 4. Train loss for GCNs, 9 classes

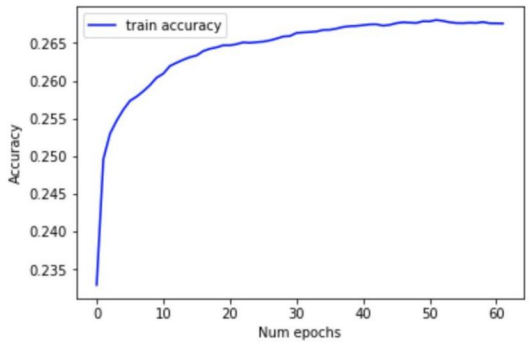

Figure 5. Training accuracy for GCNs, 9 classes

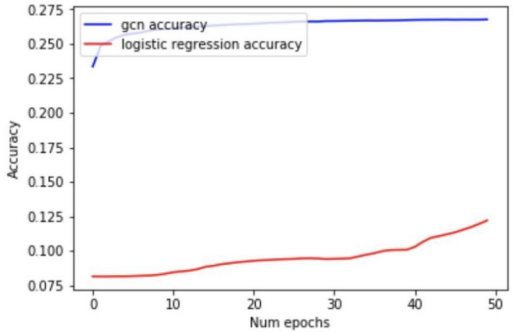

Figure 5. Comparison of training accuracy for logistic regression and GCNs, 9 classes, in the first 50 epochs

9 Conclusion

We can see that GCNs are giving better results than our current baseline models. Linear regression is performing the worst, so the problem isn't suited as a regression one. Still, the accuracy is not as high as one would it expect. One reason could be that better initial features should be used as input to the model, for both users and businesses. Nonetheless, the model is promising and should be explored further.

10 Discussion & Further Work

Since the model is not performing as well as expected, several options could be explored further:

1. Better input features for both businesses and users. Right now, we are using information that is general about both businesses and users. It could be that averaging over this information is simply not meaningful. For instance, for each

business x, take as input feature a concatenation of $[x_{image}, x_{reviews}, x_{meta}]$ where x_image is an average of the features of the last N images posted by users for that business, $x_{reviews}$ is an average of the last M reviews that business got and x_{meta} is the features vector containing metadata we are currently using. For each image, we can get a feature by passing it through a VGG16 network and taking the last feature vector. For each review, we can pass it through a Bi-LSTM layer and take the concatenation of the hidden layers as feature vector. We could find similar features for the users, based on the reviews they gave

2. Formulate this problem as a weighted graph, where the weight between an user and a business is that user's rating of the business. Right now, we are assuming that users that rated a business > 3 stars liked that business, so we are aggregating their information. It could be useful to also aggregate information from people who didn't like the business and gave negative scores.
3. Train the model longer - the model took long to train on a CPU, so if given more resources (like a GPU), could be let to run longer. Right now, we only ran GCNs for 50 epochs, but logistic regression converged around a couple hundred epochs, so it would be worth just let the model run until convergence.

11 Code Repo

The code can be found publicly available at: https://github.com/aistrate1/yelp_challenge
This is a Jupyter notebook in which I did all my work, but I also uploaded a pdf with the cells outputted.

References

[1] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

[2] Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." *Advances in Neural Information Processing Systems*. 2017.

[3] Ying, Rex, et al. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems." *arXiv preprint arXiv:1806.01973* (2018).

[4] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.

[5] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.