Personalized Recipe Recommendation Using Heterogeneous Graphs

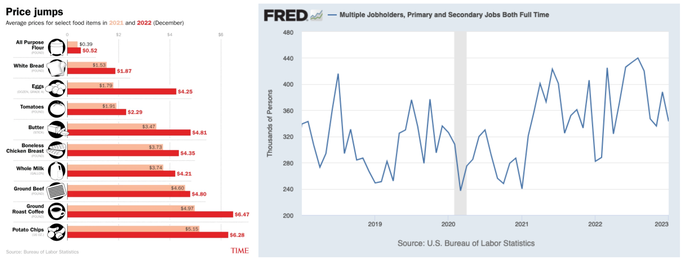
Nicholas B. DeGroot

Halıcıoğlu Data Science Institute, UC San Diego

Abstract

- We propose a system that can automate the process of meal planning and grocery shopping.
- TigerGraph is used to store and query heterogeneous graph data, which is then used to generate personalized recipe recommendations.
- We experiment with several ways to model heterogeneous graph data using the LightGCN architecture, finding that pure latent features perform best.

Motivation



Left: Increase in Average Price of Groceries
Right: Increase in Multiple Full Time Job Holders

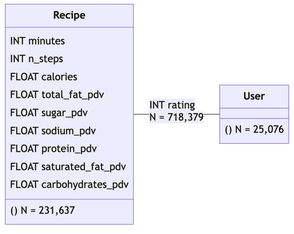
This project was born from some recent insights in the economy:

- The average cost of groceries for a household has risen by 13.5% over the last year.
- Roughly 388,000 Americans are currently working more than two **full time** jobs (nearly 5% of the workforce).

In essence, people can't afford to eat out due to budget constraints, but can't afford to eat at home due to time constraints. To this end, we propose a system that can automate the process of meal planning and grocery shopping. By automatically generating meal plans custom tailored to the users taste preferences, users save the time previously spent planning meals and the money previously spent eating out.

Data Modeling

The data for this project was sourced from `food.com` in the form of recipes and reviews. After downloading and extracting the data, we modeled and stored the data in a graph database using the TigerGraph platform. Using TigerGraph allowed us to easily model the data as a heterogeneous graph, which is a graph where nodes can be stored as different types. The graph database schema is shown below.



For the purpose of this project, we exported the data from TigerGraph in CSV format to our local machine. This was then imported using standard python libraries.

Collaborative Filtering

Our first non-trivial attempt at recommending recipes to users was to use a collaborative filtering model. Collaborative filtering is a classical algorithm that recommends items to users under the assumption that users with similar tastes will like similar items.

The model works by first computing a similarity score between users. We chose to use the pearson correlation coefficient as our similarity metric, due to its ability to handle non-centered ratings. The model then ranks each recipe based on the the similarity of users who've rated the recipe and the rating they gave it.

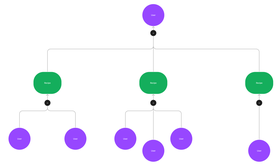
$$\text{Recipe Score} = \sum_{u'} \text{Similarity}(u, u') \cdot (\text{Rating}(u', r) - \mu_{u'})$$

One huge benefit of this model is that it can be run on-demand inside TigerGraph using GSQL. This allows for easy integration with systems that need to make recommendations in real-time.

Graph Neural Networks

Our second model attempted to use a graph neural network to recommend recipes to users. Graph neural networks are a relatively new class of neural network that are designed to make use of the structure of a graph.

Our first attempt at using a graph neural network was to build off the **LightGCN** architecture. LightGCN is a graph neural network built specifically for recommendation tasks. It works by taking one learned layer (the node embeddings), and propagating it through the graph using a normalized sum.

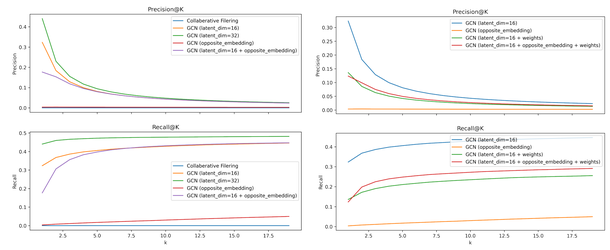


The main problem with using the LightGCN architecture is that it is not designed to handle heterogeneous graphs. For the model embeddings to be propagated, the graph must have the same dimensionality for all nodes. To solve this, we experimented with three different methods of handling the heterogeneous graph:

- **Latent Features**: The simplest method is to discard any node information and learn embeddings through latent features. This method is analogous to the original implementation.
- **Opposite Embedding**: The second method is to learn embeddings for the (normalized) opposite node's features. This method can be thought of as learning the node's affinity for those specific features.
- **Combination**: The third method is to combine the two previous methods.

For each method, we trained a model for 2000 epochs on the train set using bayesian personalized ranking loss. The best model was chosen based on precision@10 against a pre-split evaluation set.

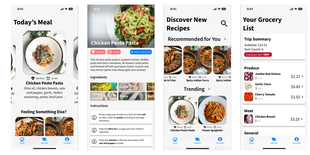
Comparison of Methods



- **Collaborative Filtering**: Could not accurately recommend recipes to users. A closer look revealed that the model was heavily reliant on the test items appearing in the training history, which was wasn't the case for most of the pre-split set.
- **GCN w/ Latent Features**: The model was able to recommend recipes to users extremely well, as suggested by the original paper.
- **GCN w/ Opposite Embedding**: The model was not able to recommend recipes to users very well. This may be a result of the smaller number of embedding dimensions.
- **GCN w/ Combination**: This model performed slightly *worse* then just using latent features, suggesting that the LightGCN architecture has a hard time learning non-latent features.

We additionally experimented with weighting the aggregations by the normalized user rating. Surprisingly, this had a extreme *negative* effect of model performance.

Future Work



Current UX Mockup

We plan to continue this work in a commercial setting. We believe that this project can be a core component of a full system that helps people save time and money by automating the process of meal planning and grocery shopping.

Future work includes:

- **User Interface**: Performing a user study to determine the best way to present the recommendations to the user.
- **Software Development**: Developing the core app to be used by the user.
- **Recommendation Improvements**: Experimenting with different models to improve the recommendations.

Website: `recipe.nickthegroot.com`
Code: `github.com/nickthegroot/recipe-recommendation`
Paper available by scanning the QR code